A SPATIAL DECISION SUPPORT SYSTEM FOR PIPE BREAK SUSCEPTIBILITY ANALYSIS AND IMPACT ASSESSMENT OF MUNICIPAL WATER DISTRIBUTION SYSTEMS

S.A. Sinske B.Eng. (Civil), M.Sc. (Geography and Environmental Study)

Dissertation approved for the degree of Doctor of Natural Sciences at the University of Stellenbosch

Promoter: Prof H.L. Zietsman

March 2002

DECLARATION

I, the undersigned, hereby declare that the work contained in this dissertation is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

ABSTRACT

Municipal water distribution maintenance is very important for sustainable urban development. Water pipe breaks result not only in a disruption in service but also in significant loss of water, which otherwise could have been sold to the consumer. In countries where water is scarce, such as South Africa, water losses can be detrimental to the living standard of people. Water pipe breaks can furthermore cause extensive damage to nearby lower-lying properties.

Existing decision support systems available in the field of water distribution system maintenance are mainly focused on leak detection and pipe rehabilitation/replacement strategy. These existing systems, however, do not address the actual causes of pipe breaks and pipe break impact is also not supported.

The aim of this research is to develop a spatial decision support system (SDSS) for pipe break susceptibility analysis and impact assessment. The engineer (or public works administrator) can apply the SDSS to model the complex pipe break phenomena in the municipal water distribution system. The SDSS can identify pipes susceptible to breaking and pipes with potentially high break impact as far as water loss and damage caused to nearby property are concerned. This combined pipe break susceptibility analysis and potential impact assessment should promote more informed decision-making on preventative maintenance measures to be taken and their prioritisation.

The dissertation consists of five parts. In the first part (Chapters 1-4) theories on information systems, fuzzy logic, object-oriented modelling, Unified Modelling Language (UML) and pipe break causes are presented. This literature review provides a basis on which the SDSS for pipe break susceptibility analysis and impact assessment can be developed.

In the second part (Chapter 5) the general user requirements and design of the SDSS are given. The general SDSS architecture, the general system

functionality and the user interface are described and designed in this part of the dissertation.

The third part (Chapter 6) provides the detailed user requirements and design of the subsystems of the SDSS. Specialised functionality for pipe break susceptibility analysis and impact assessment is added to the general design of the SDSS. Subsystems are designed for analysing the pipe break susceptibility due to age, air-pocket formation and tree-root attack. Pipe break impact assessment subsystems are also designed for assessing water loss and potential damage caused to nearby property. Finally, a combined analysis subsystem is designed for combined pipe break susceptibility analysis and impact assessment.

In the fourth part (Chapter 7), the SDSS is applied to the water distribution system of the Paarl Municipality to identify pipes in the network that have both high break susceptibility and also high break impact. The pipe break susceptibility analysis model of the SDSS is also tested and calibrated by comparing the model results with actual pipe break occurrence data of the study area.

The final chapter (Chapter 8) contains the summary and recommendations regarding the functionality of the newly developed SDSS.

OPSOMMING

Die instandhouding van munisipale waterverspreidingstelsels is uiters belangrik vir volhoubare stedelike ontwikkeling. Waterpypbreuke lei nie alleenlik tot onderbreking in diensverskaffing nie, maar ook tot beduidende waterverlies en verlies aan inkomste uit waterverkope. In lande waar water skaars is, soos in Suid-Afrika, kan waterverliese die lewenstandaard van die bevolking nadelig beïnvloed. Waterpypbreuke kan ook groot skade aan naby-geleë laag-liggende eiendomme aanrig.

Besluitnemingstelsels tans beskikbaar op die gebied van instandhouding van waterverspreidingstelsels is hoofsaaklik gerig op lekkasie-opsporing en pyprehabilitasie- en pypvervangingstrategieë. Hierdie bestaande stelsels spreek egter nie die eintlike oorsake van pypbreuke aan nie, daar word ook nie op die impak van pypbreuke ingegaan nie.

Die doelwit van hierdie navorsing is om 'n ruimtelike besluitnemingstelsel (RBS) vir pypbreuk-risiko-analise en impakberaming te ontwikkel. Die ingenieur (of stelselbestuurder) kan met behulp van die RBS die komplekse pypbreuk-verskynsel in 'n munisipale waterverspreidingstelsel modelleer. Die RBS kan pype met hoë breek-potensiaal identifiseer asook pype wat, indien dit breek, groot waterverlies of skade aan naby-geleë eiendomme sal veroorsaak. Hierdie gekombineerde pypbreuk-risiko-analise en impakberaming behoort meer oordeelkundige besluitneming te bevorder deur beter prioritisering van voorkomende instandhoudingsmaatreëls en die uitvoering daarvan.

Die proefskrif bestaan uit vyf dele. In die eerste deel (Hoofstukke 1-4) word die teorieë oor inligtingstelsels, 'fuzzy logic', objek-georiënteerde modellering, 'unified modelling language (UML)' en die oorsake van pypbreuke behandel. Hierdie literatuurstudie skep die basis waaruit die RBS vir pypbreuk-risikobepaling en impakberaming ontwikkel sal word.

In die tweede deel (Hoofstuk 5) word die algemene gebruikersbehoeftes en die ontwerp van die RBS uiteengesit. Die algemene RBS struktuur en die gebruikerskoppelvlak word in hierdie deel van die proefskrif beskryf en ontwerp.

In die derde deel (Hoofstuk 6) word die gedetailleerde gebruikersbehoeftes en die ontwerp van die substelsels van die RBS uiteengesit. Gespesialiseerde funksionaliteit vir pypbreuk-risikobepaling en impakberaming is tot die algemene ontwerp van die RBS bygevoeg. Substelsels is ontwerp vir die ontleding van pypbreuk-risiko as gevolg van ouderdom, lugblaas-vorming en boomwortel-aanval. Substelsels vir impakberaming is ook ontwerp om waterverlies en potensiële skade aan eiendomme vas te stel. Ten slotte word 'n gekombineerde ontledingsubstelsel vir gekombineerde pypbreuk-risikobepaling en impakberaming opgestel.

In die vierde deel (Hoofstuk 7) word die RBS toegepas op die waterverspreidingstelsel van die Paarlse munisipaliteit om pype uit te ken wat beide 'n hoë breuk-risiko en 'n hoë breuk-impak bevat. Die pypbreuk-analise model van die RBS is ook getoets en gekalibreer deur die resultate van die model te vergelyk met data van werklike pypbreuke in die studiegebied.

Die laaste hoofstuk (Hoofstuk 8) bevat die samevatting en die aanbevelings rakende die funksionaliteit van die voorgestelde RBS.

ACKNOWLEDGEMENTS

I wish to thank:

- Prof HL Zietsman for all his assistance in finalising the dissertation.

- The Paarl municipality for providing me with the data used in this dissertation.

CONTENTS

FIGURES

TABLES

ACRONYMS

CASE	Computer-Aided Software Engineering
DBMS	Data Base Management System
DDM	Dialog, Data and Modelling paradigm
DGMS	Dialog Generation and Management System
DLL	Dynamic Link Library
DOF	Degree of Fulfilment
DSS	Decision Support System
EDP	Electronic Data Processing
EIS	Executive Information System
ESRI	Environmental Systems Research Institute
FAM	Fuzzy Associative Memory
GIS	Geographical Information System
GPS	Global Positioning System
GUI	Graphical User Interface
MBMS	Model Base Management System
MFEP	Multifactor Evaluation Process
MIS	Management Information System
MSS	Management Support System
SDSS	Spatial Decision Support System
UFW	Unaccounted-for Water
UML	Unified Modelling Language
VBA	Visual Basic for Applications

CHAPTER 1: MUNICIPAL WATER DISTRIBUTION SYSTEM MAINTENANCE FOR SUSTAINABLE URBAN DEVELOPMENT

Civil infrastructure systems have a direct impact on the economic growth of a country and its ability to compete world-wide. If the civil infrastructure is not maintained properly, it cannot support the level of service for which the facilities were originally designed. Inadequate maintenance and insufficient use of these infrastructure facilities reduce both the capacity of the infrastructure system and the strength of the national economy (Shen & Grivas 1996). In this chapter the importance of specifically the municipal water distribution system for sustainable urban development will be discussed.

1.1 BACKGROUND

Of all civil infrastructure, the water distribution system is believed to be the most essential, since it is impossible to live or work in a community if there is not potable water available. Various industries and businesses can also not operate without a well functioning and reliable water pipe network. The water pipe network is indeed the lifeline of each community (Böhm 1993); it plays a significant role in the public health of a community and can become a determinant of the community's growth path.

Water loss in a water distribution system as the result of neglected maintenance may place a significant burden on the municipality, since it contributes to the 'unaccounted-for water' (UFW) of the system. UFW consists mainly of water losses incurred by burst pipes and leakage; malfunctioning water metres that do not register the flow correctly and illegal connections may also contribute to large amounts of UFW. The efficiency of the system can be expressed in terms of the unaccounted-for water (UFW) index. This index is simply the ratio of the measured consumption to the measured supply from the reservoir. A water distribution system with a UFW of 10% and lower is regarded as normal, whereas a UFW of 30% and higher is very inefficient.

An UFW index of 50% is not uncommon in some developing countries. Even in many developed countries a poor maintenance strategy can account for a UFW of 25% or more (UNESCO 2000). When the water supply system of a community must be extended to meet future demands, it is imperative to first reduce UFW% to an acceptable level before extending the capacity of the system components. By doing so, future capital expenditure could be restricted and prioritised in areas of higher needs. Furthermore, operating profits of the water department would improve, enabling easier debt repayment on capital expenditure (Powell, Constantinides & Kolovopoulos 1996).

Apart from the economic savings that can be obtained by reducing the system losses, the saving of water in a water scarce country is of paramount importance for sustainable growth. The margin between the global available water resource and the volume of water used is going to diminish in the future, mainly as the result of population growth. By the year 2025 the regions of stress with regard to water shortage will be extended to include about two thirds of the world's population. By 2050 they will probably cover most of the globe. As the crisis approaches and as water resources become scarcer, the risk of conflict over them will become greater. After 2025 climatic change could also aggravate conditions if precipitation amounts decrease in the major food-producing regions and evaporation rates increase. More water will then be needed for irrigation, which will further increase the conflict over the water resources, since the growing conurbations will also have increasing water demands. The cost of water may rise because of this competition and inflate food prices (UNESCO 2000).

According to the World Commission on Environment and Development, about 80 countries with 40% of the world's population already suffer from severe water shortages. South Africa is no exception to this global problem and can probably be regarded as being amongst the worst stressed countries, in terms of water resources, due to its unfavourable hydrological characteristics (Water Research Commission 2000). South Africa lies in the drought belt of the globe, and may therefore be regarded as a dry country (Department of Water

Affairs 1986; Roberts 1994). The groundwater reserves are scarce and sometimes also saline. The surface water resources are limited – total runoff is estimated at a mere 53 500 million cubic metres per annum. Its irregularity necessitates the construction of substantial storage facilities. The existing large dams in South Africa create a total storage volume of about half the total average runoff. This figure would be relatively large for countries in temperate zones of the world, but still insufficient for South Africa (Van Robbroeck 1994; Water Research Commission 2000). The storage dams also become silted up and precious water is lost through evaporation and pollution. In 1970 already a Commission of Enquiry into Water Matters warned that unless South Africa develops, manages and utilises the water resources properly, serious water shortages may occur by the year 2000 and onwards (Roberts 1994). The rapidly growing population further requires a vigorous economy and places heavy demands on the water resources for power generation, mining, industry and agriculture. Improving social conditions needs large quantities of water for domestic purposes and for sanitation (Van Robbroeck 1994). Increasing population and improved living standards will further lead to a growing water demand and, unfortunately, also to more pollution, which will reduce the availability of water. As water shortages and needs will increase in South Africa, competition for water amongst all sectors (domestic, industrial, ecological and agricultural) will also become more intense. Despite huge investments, the developed water supply infrastructure has not met the demands due to inadequacies, resource mismanagement and past policies, which have resulted in half of the population not having access to an adequate water supply. Although the domestic or local government usage of water makes up only 12% of the total water usage in the country, it is also the most expensive type of water that is provided. There is still concern about this highly treated and costly resource of South Africa that is being used inefficiently by consumers, with high water losses in the system as well, still posing a problem (Water Research Commission 2000).

Cape Town and the surrounding areas currently experience a severe water shortage. The present water supply can only meet the water demand by imposing water restrictions. The present supply of 475 million cubic metres per annum cannot be augmented significantly prior to the year 2004 when the Skuifraam Dam will come into operation. This means that an even more rigorous water saving programme must be implemented (Department of Water Affairs and Forestry 1999). In addition to water restrictions, system losses (as the result of pipe breaks and leakage) in the water distribution systems of the Western Cape will have to be significantly reduced.

Apart from the water loss already mentioned, pipe breaks in a water distribution system as the result of neglected maintenance can cause extensive damage to property. The water from a pipe break is usually coloured brown from all the filling material that is swept with the flow. There have been numerous reports of this brownish water causing damage to swimming pools and also penetrating low-lying houses and ruining precious carpets and furniture. There are also reports of structural damage caused by the streaming water from a pipe break, which led to the collapse of solid brick walls (*Eikestadnuus* 1999; *Eikestadnuus* 2001a; *Eikestadnuus* 2001b).

Pipe breaks can also cause extensive disruptions in the water supply service, which is not only inconvenient but can also lead to financial losses for certain businesses and industries that rely on an uninterrupted water supply. Damage to roads, a reduction in fire-fighting capacity and huge pipe repair or replacement costs are some other detrimental pipe break impacts (Clark, Stafford & Goodrich 1982; Böhm 1993). The pipe repair costs, consisting of hourly wages and material costs, can negatively influence the municipal budget if pipe breaks begin to escalate. The disruption in service is inconvenient for the consumer since in the beginning, when the break has not yet been reported, there is very little water pressure, and sometimes no water at all, and then during the repair period the water is shut off completely. In cases where the shut-off valves are placed too far apart or unsuitably placed in the network, many stands will then be without water during the pipe repair period.

Causes of pipe breaks are usually the result of a complex combination of factors, each contributing to a certain extent to the deterioration and eventual breakage of a pipe (Morris 1967; Kottmann 1978; Ciottoni 1983; O'Day 1983). The factors involved are often difficult to quantify and the relationships between the factors involved are often not precisely clear. In pipe break susceptibility analysis complex models should be applied in which all these interrelated pipe-breaking factors are taken into account. Pipe break impact assessment, especially the assessment of water loss and the potential damage caused to property, requires similar complex modelling.

Currently there are many decision support systems available in the field of water distribution system maintenance for leak detection, water meter management and pipe rehabilitation/replacement strategy. The latter is an economical analysis tool to decide whether a pipe should be rehabilitated or replaced and is based on the statistical analysis of break occurrence data (which is not always available). These existing systems, however, rely on statistical methods only and do not address the actual pipe break causes. Pipe break impact assessment is also not supported by these systems.

1.2 RESEARCH OBJECTIVES

The aim of this research is to develop a spatial decision support system (SDSS) for pipe break susceptibility analysis and impact assessment. The engineer (or public works administrator) can apply the SDSS to model the complex pipe break phenomena in the municipal water distribution system. The SDSS can identify pipes susceptible to breaking and pipes with potentially high break impact as far as water loss and damage caused to nearby property are concerned. This combined pipe break susceptibility analysis and potential impact assessment should promote more informed decision-making on preventative maintenance measures to be taken and their prioritisation. Preventative maintenance can then be focused on locations pointed out by the SDSS so to be able to prevent excessive water loss and damage to property. Newly planned water distribution systems can also be tested by the SDSS to identify pipes with high break susceptibility and potentially high break impact. Designs can then be altered or improved before commencing with the actual construction.

The main objective of this dissertation is to develop a SDSS that should basically support the following:

- Assist the engineer (or the public works administrator) in understanding the complex pipe break phenomena in the municipal water distribution system;

- Identify pipes in the water distribution system that have high break susceptibility;

- Identify pipes in the water distribution system that have high break impact with regard to water loss;

- Identify pipes in the water distribution system that have high break impact with regard to property damage;

- Conduct a combined pipe break susceptibility analysis and impact assessment. Hereby different combinations can be analysed such as identifying pipes that have both high pipe break susceptibility and potentially high break impact (as the result of water loss only). Or identifying pipes that have both high pipe break susceptibility and potentially high break impact (as the result of property damage only). Or identifying pipes that have both high pipe break susceptibility and potentially high break impact (as the result of both water loss and property damage).

The secondary objectives to meet the main objectives are the following:

- The SDSS will be based on the concepts of information systems theory and object-oriented modelling. The core functionality of the system will be written in Visual Basic for Applications (VBA) so that it can be fully integrated with a modern geographical information system (GIS) that has a built-in VBA Editor. Some functionality will be added via external programs which will be linked to the SDSS:

  - A fuzzy-logic-based controller (defined later in Chapter 2) will be linked to the system to supply the fuzzy logic analysis functionality that is needed for the modelling of complex aspects regarding pipe break susceptibility analysis and impact assessment.

  - A water distribution system analysis program will be linked to the system for performing intricate network flow operations that are needed to determine the outflow rate from a pipe break.

  - An interpolation program will be linked to the system, which will be used to interpolate elevation points needed in the modelling and assessment of the potential damage caused to property.

  - A spreadsheet will also be linked with the SDSS to draw graphs of the analyses results.

- The Unified Modelling Language (UML) will be used to document all phases of the requirements analysis and design of the SDSS. This standardised documentation approach should promote communication in subsequent system development work.

- The SDSS will be applied to the water distribution system of Paarl, a medium sized town in South Africa for which relevant data was available. Pipes will be identified with high break susceptibility and potentially high break impact (regarding water loss and damage caused to property). The output results from the pipe break susceptibility analysis will then be compared with actual break occurrences in the network and adjustments will be made to the initial parameters and modelling rules until the system is calibrated (trained). The results and new information (viz. the decision rules) obtained from the trained system can then be incorporated into the knowledge base of the SDSS to be applied later also to other water distribution systems (also newly designed systems).

1.3 RESEARCH FRAMEWORK

This dissertation is structured as follows (see Figure 1.1):

- Chapter 1 is an introductory chapter on the importance of municipal water distribution system maintenance for sustainable urban development. The chapter also contains the objectives of the dissertation and the literature survey.

- Chapter 2 is an overview of information systems. The concepts of a system, the different types of computer-based information systems and information systems development methodologies are discussed. The fuzzy-logic-based controller, a special type of computer-based information system, is also described in detail in this chapter, since it will be linked with the SDSS to conduct complex analyses.

- In Chapter 3 object-oriented modelling concepts are discussed. The use of the Unified Modelling Language (UML) diagrams is also described. UML is used in this dissertation to document the user requirements and system designs.

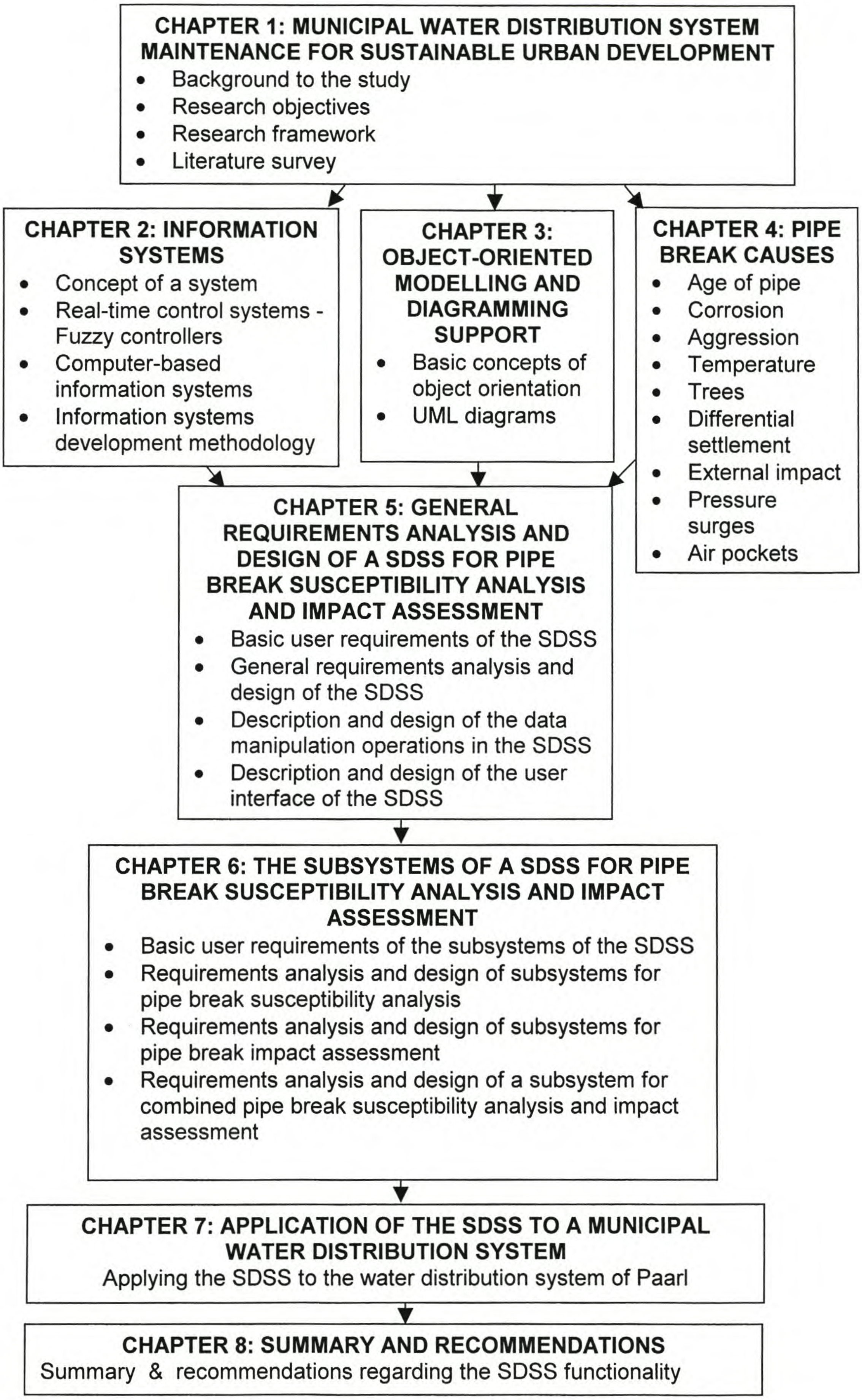
- Chapter 4 is a literature study of known pipe break causes. Many of these pipe break causes will be incorporated in the knowledge base of the SDSS. Since Chapters 2, 3 and 4 are of equal importance to the development of the SDSS, they have been placed on the same level in Figure 1.1.

- Chapter 5 contains the general user requirements and design of the SDSS. The general SDSS architecture, the general system functionality and the user interface are described and designed.

- Chapter 6 provides the detailed requirements and design of the subsystems of the SDSS. Specialised functionality for pipe break susceptibility analysis and impact assessment is added to the general design of the SDSS.

- In Chapter 7 the SDSS is applied to the Paarl water distribution system. The SDSS is also tested and calibrated by comparing model results with actual pipe break occurrence data of the study area.

- Chapter 8 contains the summary and recommendations regarding the functionality of the newly developed SDSS.



CHAPTER 1: MUNICIPAL WATER DISTRIBUTION SYSTEM MAINTENANCE FOR SUSTAINABLE URBAN DEVELOPMENT

- Background to the study
- Research objectives
- Research framework
- Literature survey

CHAPTER 2: INFORMATION SYSTEMS

- Concept of a system
- Real-time control systems - Fuzzy controllers
- Computer-based information systems
- Information systems development methodology

CHAPTER 3: OBJECT-ORIENTED MODELLING AND DIAGRAMMING SUPPORT

- Basic concepts of object orientation
- UML diagrams

CHAPTER 4: PIPE BREAK CAUSES

- Age of pipe
- Corrosion
- Aggression
- Temperature
- Trees
- Differential settlement
- External impact
- Pressure surges
- Air pockets

CHAPTER 5: GENERAL REQUIREMENTS ANALYSIS AND DESIGN OF A SDSS FOR PIPE BREAK SUSCEPTIBILITY ANALYSIS AND IMPACT ASSESSMENT

- Basic user requirements of the SDSS
- General requirements analysis and design of the SDSS
- Description and design of the data manipulation operations in the SDSS
- Description and design of the user interface of the SDSS

CHAPTER 6: THE SUBSYSTEMS OF A SDSS FOR PIPE BREAK SUSCEPTIBILITY ANALYSIS AND IMPACT ASSESSMENT

- Basic user requirements of the subsystems of the SDSS
- Requirements analysis and design of subsystems for pipe break susceptibility analysis
- Requirements analysis and design of subsystems for pipe break impact assessment
- Requirements analysis and design of a subsystem for combined pipe break susceptibility analysis and impact assessment

CHAPTER 7: APPLICATION OF THE SDSS TO A MUNICIPAL WATER DISTRIBUTION SYSTEM

Applying the SDSS to the water distribution system of Paarl

CHAPTER 8: SUMMARY AND RECOMMENDATIONS

Summary & recommendations regarding the SDSS functionality

FIGURE 1.1 Research framework

1.4 LITERATURE SURVEY

Literature surveys on four main topics were conducted for this dissertation. The availability of water (globally and specifically in South Africa) was researched to justify the development of the water loss assessment subsystem of the SDSS. Information systems (including decision support systems and information systems development methodologies) were studied, because the SDSS to be developed in this dissertation will be based on these existing information systems. Object-oriented modelling and diagramming support techniques were reviewed, since they will be applied in all system analysis and design phases. An extensive literature survey was conducted on pipe break causes to gain information on this specialised field of civil engineering. The pipe break susceptibility analysis models of the SDSS will largely be based on this information. The literature surveys on these four main topics will further be discussed in the sections that follow.

1.4.1 Availability of water

A literature survey was conducted to gain information on the global availability of water, the availability of water in South Africa and also specifically in the Western Cape. The survey was conducted to justify the development of the water loss subsystem of the SDSS for reducing water loss as the result of pipe breaks in a water distribution system. The source of information for the global availability of water mainly stems from the Internet web site of UNESCO where information on global water matters is updated regularly (UNESCO 2000). The Water Research Commission reports (Water Research Commission 2000) have mainly been used as source of information to assess the availability of water in South Africa. The Internet web site of the Department of Water Affairs and Forestry provides an overview of the water situation in Cape Town and surrounding areas (Department of Water Affairs and Forestry 1999). The web site contains the water plan for the Cape Metro. Water demand projections are given and possible water supply augmentation schemes are presented.

1.4.2 Information systems

The spatial decision support system to be developed in this dissertation is a special type of decision support system, which again is a special type of information system. Decision support systems and information systems are also special types of systems. A literature survey has been conducted on these four topics, viz. systems theory, information systems (including information systems development methodology), decision support systems and spatial decision support systems. The classical texts of Von Bertalanffy (1968), Checkland (1981), as well as Flood and Carson (1988) have been reviewed. Bennett, McRobb and Farmer (1999) provide an extensive account on the related subjects of systems and information systems. The authors also refer to various classical texts. The more recent work of Checkland (1999) and Avison and Fitzgerald (1999) on information systems development, as well as the work of Srinivasan *et al.* (2000) on decision support systems, has also been included in the literature survey. Engineering journals (such as *Transportation Research*) and Internet sources have been reviewed to find the latest information and developments in spatial decision support systems.

The SDSS is linked to a fuzzy controller for conducting complex analyses. Fuzzy controllers are highly specialised information systems on which a detailed survey had to be conducted, as well as on fuzzy logic, which is the theory on which fuzzy controllers are based. The original work of Zadeh (1965), the founder of fuzzy logic, was reviewed. The work of other well-known authors in this field, such as Klir *et al.* (1995) and Chang (1997), was also included in the literature survey. Fuzzy logic is used in many modern electronic and mechanical devices and there are major developments currently in fuzzy logic software. Scientific magazines (such as *GEOEurope*) and Internet sources have therefore also been reviewed to find the latest information and developments in fuzzy logic.

1.4.3 Object-oriented modelling and diagramming support

The concepts of object orientation are discussed thoroughly in Caspers (1994), Sodhi *et al.* (1998), and Bennett, McRobb and Farmer (1999). In the latter, diagramming support using the Unified Modelling Language (UML) is discussed in depth. The work of Booch, Rumbaugh and Jacobson (1999), the developers of UML, has also been studied in detail.

The field of object-oriented modelling and diagramming with Computer-Aided Software Engineering (CASE) tools is very dynamic, and it is believed that major developments in this field can be expected in the future.

1.4.4 Pipe break causes

A literature survey was conducted to gain information on known pipe break causes. The information stems mainly from North American and European pipe break studies. Some of the pipe break causes may not be relevant to South African water distribution systems, but are also presented in this dissertation so that the system could later be extended to support pipe break preventative maintenance in those regions as well.

The study of the causes of pipe breaks is a highly specialised field of civil engineering. There are very few books available on this topic. Böhm (1993) provides an account on water distribution system maintenance with chapters included on pipe break causes. All other information on pipe break causes in this dissertation has been obtained from journals. The literature survey conducted on pipe break causes is extensive, since the SDSS pipe break susceptibility analysis models will largely be based on this information. Only the key authors will be singled out in the discussion that follows.

The classical texts of Arnold (1960) and Morris (1967) on pipe break causes have been reviewed. Most of these causes are still very relevant today. O'Day (1983) gives a detailed account on the analysis of water pipe network infrastructure conditions with the focus on pipe age.

The classical texts of Kalinske and Bliss (1943), Lescovich (1972), Wisner *et al.* (1975) and Krug (1988) on air pockets in water pipes were reviewed. The various studies conducted by Kottmann in the period 1980 to 1995 on entrapped air-pockets as a cause of pipe bursts and the work of Böhm (1993) were the major sources of information for airbursts in this dissertation. Catalogues of the international air-valve manufacturing company Vent-O-Mat and the South African pipe company AC Pipes, containing valuable information on pipeline design and also information on the causes of airbursts, have also been reviewed.

The *Journal of Arboriculture* and the *American Forests* journal contain information on the conflict between trees and urban infrastructure. Additional information on pipe breakage as the result of tree roots was obtained from the Stellenbosch and the Paarl municipalities. The latter also kindly made available the digital pipe break occurrences data for this dissertation.

Finally, valuable practical information was obtained from various site visits where pipe break maintenance work has been conducted on the Stellenbosch water distribution system. The photographs in Chapters 4 and 6 were taken during these site visits.

1.5 SUMMARY

In this chapter the importance of municipal water distribution system maintenance for sustainable urban development has been conveyed. Water loss in water distribution systems resulting from pipe breaks should be reduced, since it has been found that the water demand in South Africa (and in many other countries throughout the world) will be difficult to meet in the near future. The research objectives, research framework and literature survey have also been presented. The next chapter will give an overview of information systems, to which spatial decision support systems also belong.

CHAPTER 2: INFORMATION SYSTEMS

The spatial decision support system (SDSS) for pipe break susceptibility analysis and impact assessment that will be developed in this dissertation is a particular type of information system. This chapter will firstly introduce the general concepts of a system on which the SDSS will fundamentally be based. The different types of information systems will then be discussed. Finally, a methodology will be presented to develop information systems successfully so that the SDSS will comply with the requirements of a municipality and thus be able to support the preventative maintenance planning of a water distribution system.

2.1 THE CONCEPT OF A SYSTEM

A system is generally described as an assembly of elements related in an organised whole (Flood & Carson 1988). A system, possibly decomposed into a collection of subsystems, is a set of elements organised to accomplish a purpose and can be described by a set of models from different viewpoints (Booch, Rumbaugh & Jacobson 1999). Furthermore, systems always have some kind of goal, even if it is only to help understand some real-world situation. The process of systems thinking, i.e. consciously organised thinking using systems ideas, can also support the development of successful information systems (Bennett, McRobb & Farmer 1999; Checkland 1999).

The origin and evolution of system theory, the system taxonomy, and the various system characteristics and components will be discussed in the sections that follow.

2.1.1 The origin and evolution of system theory

Systems ideas emerged as a generalisation of ideas about organisms, which were developed within the discipline of biology in the first half of the twentieth century. The reductionist approach of scientific methods failed to explain the complex processes and interactions within organisms, since an organism has highly organised structures that cannot be explained by studying only its constituent parts. The school of thought associated with the so-called 'organismic biologists' was focused upon the organism as the unit of analysis in biology, and developed ideas about the processes which characterise metabolism and self-reproduction in organisms. It was one of these organismic biologists, the Austrian Ludwig von Bertalanffy, who founded the systems movement in the late 1940s. In his work entitled *General System Theory* he argues that these ideas about organisms could be extended to complex wholes of any kind i.e. to 'systems'. Bertalanffy's ideas were taken up by the economist K. E. Boulding, the physiologist R. W. Gerard, and a mathematician A. Rapoport, who founded the Society for General Systems Research in 1954. The aim of the society was to foster communication between scientists from different disciplines in order to facilitate the sharing of knowledge, and to minimise the duplication of scientific research. Their idea was that a meta-level theory and models 'applicable to more than one of the traditional departments of knowledge' could be developed. This kind of meta-level problem solving envisaged by the pioneers has, however, never fully taken place, since the reality is too complex to be modelled by a single system. The systems ideas, however, have made contributions within many different subject areas. The original ideas of the protagonists of general system theory have been vindicated to some extent in that there is now a set of systems ideas which find application in many fields (Flood & Carson 1988; Checkland 1999).

The Second World War, with its attendant problems of logistics and resource management, acted as a catalyst for the growth of systems science with the focus on 'hard' quantitative analysis (Flood & Carson 1988).

Early attempts to apply the 'hard' systems methodology in a social problem-solving context were not very successful. The main difficulties have, however, been remedied to a large extent by the 'soft' systems methodology, mainly introduced by P. Checkland in the 1970s with his work on the human activity system (see Section 2.1.2) and the systems thinking concepts. The systems thinking approach allows more flexibility when designing systems, since it is based on the notion that systems exist only in our thoughts and thus represent subjective views of reality, not the reality itself (Checkland 1981; Flood & Carson 1988; Bennett, McRobb & Farmer 1999; Checkland 1999).

The present object-oriented systems modelling era, which started mainly in the 1990s, blends the concepts of object orientation into systems theory. Hereby a complex system can be described by a set of object-oriented models, and from different viewpoints omitting detail that is not relevant to the particular perspective (Bennett, McRobb & Farmer 1999; Booch, Rumbaugh & Jacobson 1999).

2.1.2 System taxonomy

Checkland (1981) classifies systems into the following five classes: *natural systems, designed physical systems, designed abstract systems, human activity systems* and *transcendental systems.*

- A *natural system* is one that represents a real-world phenomenon, such as for example a tropical rainstorm system, the human immune system, etc.

- A *designed physical system* is similar to a natural system except that it is man-made and is the result of conscious design to fulfil some human purpose. Examples of designed physical systems are water transfer systems, sewer systems, water distribution systems, etc.

- A *designed abstract system* represents the ordered conscious product of the human mind such as mathematics, poems or philosophies.

- A *human activity system* is a system that centres on purposeful activity, such as a business, a club, a team, or any other organised activity. A human activity system is thus a study of human affairs and it includes the subject areas of management and, within that, information systems (Checkland 1999). An information system is an artificial system which is constructed to help people in a human activity system to achieve their goals. For an information system to be able to support the human activity system in a useful way, it is of the utmost importance to first understand the meaning and purpose of the human activity system.

- *Transcendental systems* are systems beyond knowledge that cannot be categorised into any of the above four known system classes.

Almost all these system classes are applicable to this dissertation. The pipe breakage phenomenon in water distribution systems can be classified as a natural system, since pipe breaks are mainly caused by natural forces acting on the water pipes either internally or externally (see Chapter 4). A water distribution system is a designed physical system to deliver water to the consumer. The impact of a breakdown in a designed physical system (such as the water loss assessment of a pipe break in a water distribution system) can in general be assessed more accurately than a breakdown in a natural system, since the latter is often too complex to be fully comprehended. The water maintenance department is a human activity system that performs active (emergency) and reactive (preventative) maintenance on the water distribution system. The SDSS to be developed in this dissertation is a special type of information system, which can eventually be used in the water maintenance department to support the maintenance planning.

2.1.3 Characteristics and components of a system

All systems share certain basic characteristics which include the following:

- systems are goal-oriented
- systems convert inputs into outputs
- systems are holistic
- systems are differentiated

- systems are hierarchical
- systems are inter-disciplinary
- systems are synergistic
- systems must be maintained
- systems exist in an environment
- systems are separated from the environment by some kind of boundary
- systems have interfaces
- systems that endure have a control mechanism

(Von Bertalanffy 1968; Emery 1969; Hall & Dracup 1970; Checkland 1981; Flood & Carson 1988; Bennett, McRobb & Farmer 1999; Checkland 1999).

The main system characteristics and components are depicted in Figure 2.1 and will now further be elaborated on. For the purpose of this discussion, they are grouped under the headings that will follow.

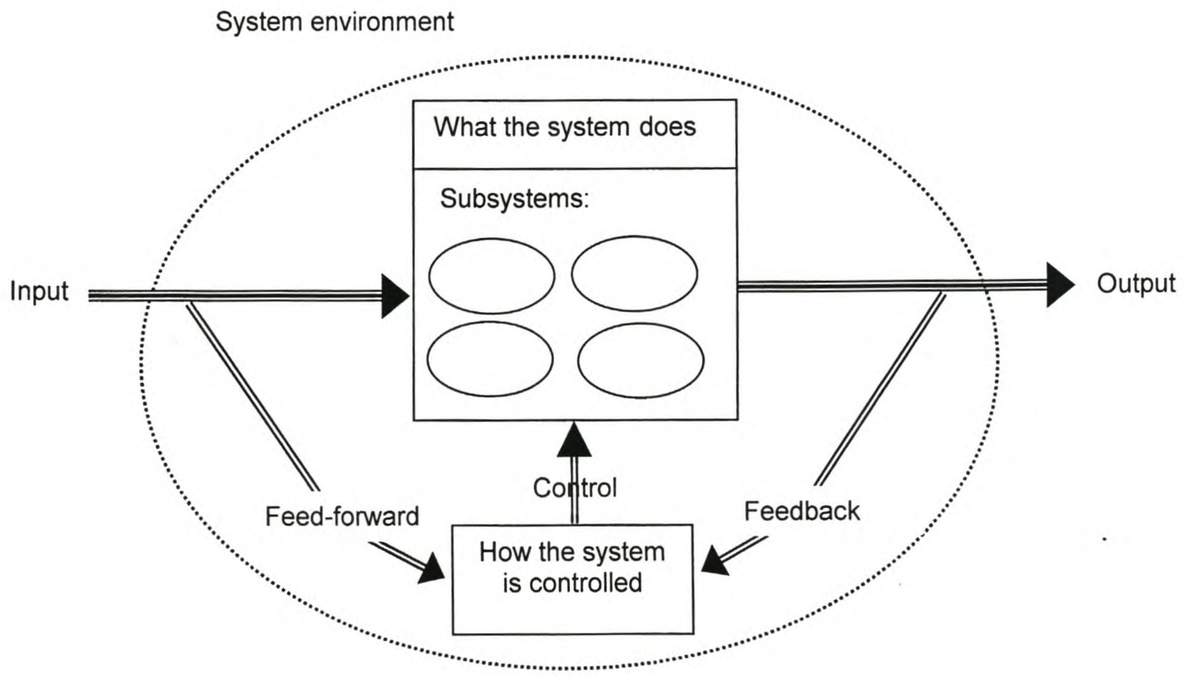


FIGURE 2.1 System characteristics and components (adapted from Bennett, McRobb & Farmer 1999)

2.1.3.1 System boundary and environment

A system exists in an environment and is separated from its environment by some kind of boundary (see Figure 2.1). A system is termed an open system if the boundary is permeable and allows inputs from and output to the environment. The system boundary has very much to do with scale and the choice of a system boundary corresponds to the subject of interest. System boundaries can overlap or coincide. Two systems may thus have the identical boundaries and yet still be distinct. Systems that share the same boundary may have different levels of detail since they represent different aspects of reality. Also, a subsystem of a particular system can simultaneously be part of another system.

2.1.3.2 Conversion of inputs to outputs

Inputs to the system originate outside the system and are taken in to be used in some way. Outputs are created by the system and are sent into the environment in order to have an effect somewhere else. An important characteristic of purposeful systems is the way they transform inputs into outputs, since this is the way they fulfil their goals (objectives). The user of the system only needs to know the relationship between the inputs and outputs. His focus will be on the interaction with the system, i.e. the input that must be provided and the results that can be obtained from the system. The user thus does not need not to know the internal workings of the system and will also have no access to it. This is called the black box approach, since it treats the system as an opaque box whose internal workings are completely hidden (see Figure 2.2).

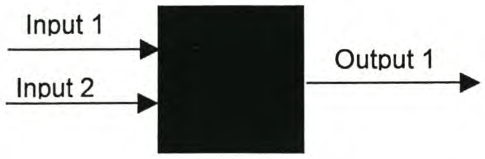


FIGURE 2.2 Black box view of a system

The main advantages of the black box approach are that it simplifies system operation and it offers a secure system that cannot be corrupted by the user (intentionally or unintentionally). The main disadvantage of the black box approach is that the user cannot customise or extend the system – he can thus only use the system as it is. Care should also be taken if the functional specifications of such a black box type of system are not clearly stipulated, since the system can then easily be used incorrectly and will then give incorrect results.

2.1.3.3 Subsystems

Most systems can be differentiated, i.e. broken down into a number of subsystems (see Figure 2.1). Each subsystem has its own function clearly differentiated from that of others. If two subsystems are not clearly differentiated - in other words, if they have similar goals and responsibilities - friction will develop between them. The only way in which this problem can be solved is either by absorbing one subsystem into the other, or by clearly defining and differentiating the functions of each. A subsystem can be seen as a system on its own, since it has certain specialised functionality, or as a subsystem that contributes to the larger system. The latter is also known as system synergy, or the holistic systems view, which implies that the whole is greater than the sum of the parts. There are thus emergent properties relating to the whole system but not necessarily present in any of the parts. Systems are often hierarchically structured, in the sense that most systems are subsystems of a super-system, and are themselves super-systems of a number of subsystems. Horizontal and vertical inter-disciplinary communication provides the necessary cohesion for these highly differentiated systems still to be synergistic. This communication takes place through shared system boundaries via an interface, so that the output from one system (or subsystem) can then simultaneously be an input in another.

2.1.3.4 Control in systems

One of the main characteristics of a system, already mentioned, is that it is goal oriented. In order to advance towards a goal, it is often necessary to check the current position and to make the necessary course correction. This monitoring and correcting are the essence of system control. Many systems have a specialist subsystem whose function is to control the operations of the system as a whole. System control is usually based on a comparison between two or more input values, whose similarity or difference guides a decision about whether any controlling action is required. Control systems use a feedback or a feed-forward loop to control the operations (see Figure 2.1). In a feedback loop one or more outputs of the system are sampled and literally fed back to the control unit, where a logic mechanism (this could be either mechanical or electronic) decides what action is required. This type of adjustment is called *negative* feedback, because it aims to maintain the system's equilibrium by opposing deviations from some norm. By contrast, positive feedback works by reinforcing deviations instead of opposing them and therefore tends to increase movement away from equilibrium. This type of system, however, is seldom used, and then usually in cases where a steady state is undesirable. Control systems can also use a feed-forward loop (see Figure 2.1). Feed-forward systems rely on sampling the system's inputs rather than its outputs. In manufacturing systems, for example, feed-forward control will adjust the manufacturing process to suit the level of incoming orders (inputs). This would avoid manufacturing products for which there is no market. Feed-forward control can thus help a system to maintain itself in an environment where there are many fluctuations.

2.2 REAL-TIME CONTROL SYSTEMS - FUZZY CONTROLLERS

Real-time systems are explicitly concerned with the direct control of a system's operations, often physical in nature. For this reason they are perhaps best considered as a control subsystem (see Section 2.1.3.4) of a physical processing system (Bennett, McRobb & Farmer 1999). Real-time control systems, which are often also computer-based, however, fulfil a very different function to the other computer-based information systems, which will be discussed later in Section 2.3, and will therefore be discussed here separately.

The logical control mechanism (see Section 2.1.3.4) in real-time control systems is often based on fuzzy logic (see Section 2.2.2) - real-time control systems are therefore also referred to as fuzzy controllers (see Section 2.2.3). A fuzzy controller will be linked to the SDSS of this dissertation (see Section 5.4.4), to model the complex aspects in pipe break susceptibility analysis and impact assessment. The fuzzy controller will, however, not be used as a classic control system to maintain equilibrium, but will only receive inputs continuously which it must convert to outputs (viz. pipe break susceptibility and impact assessment values). The feedback loop (see Section 2.1.3.4), where the output results are fed back to the controller, is thus not applicable. The feed-forward loop (see Section 2.1.3.4), however, will be used to pass on new input values to the fuzzy controller for which output values are to be calculated.

The aim of this section is to introduce fuzzy controllers and the theory they are based on, viz. fuzzy sets and fuzzy logic.

2.2.1 From ordinary crisp sets to fuzzy sets

According to the traditional view, science should strive for certainty in all its manifestations (precision, specificity, sharpness, consistency, etc.); hence, uncertainty (imprecision, non-specificity, vagueness, inconsistency, etc.) is regarded as unscientific, undesirable and something that should be avoided by all possible means (Klir & Yuan 1995; Burrough & McDonnell 1998). According to the alternative (modern) view, uncertainty is considered essential to science; it is not only an unavoidable plague, but it has, in fact, a great utility (Klir & Yuan 1995).

In general we deal with problems in terms of systems that are constructed as models of either some aspects of reality or some desirable man-made objects. The main purpose of constructing these models is to understand some phenomenon of reality, be it natural or man-made, to be able to make adequate predictions or 'retrodictions' and learning how to control the phenomenon (Klir & Yuan 1995). Conceptual models, based on fundamental mathematics, are often difficult to construct and to solve because the phenomenon they describe is usually complex, consisting of many interrelated factors. The individual factors or parameters are often difficult to measure and classify accurately and the relationships between the factors are often also ill-defined, uncertain and complex (Zadeh 1992; Chang 1997).

It is generally agreed that an important point in the evolution of the modern concept of uncertainty was the publication of a seminal paper by Dr Lotfi A. Zadeh (1965). In his paper Zadeh introduced a theory whose objects - fuzzy sets - are sets with boundaries that are not precise. The significance of Zadeh's paper was that it challenged not only probability theory as the sole agent for uncertainty, but the very foundations upon which probability theory is based: Aristotelian two-valued logic (Klir & Yuan 1995). In classical crisp set theory, based on two-valued logic, any element in the universe of discourse is either a member of a set or not a member of that set. This means that a particular element belongs fully to a particular set or does not belong to that set at all. Contrary to crisp set theory, fuzzy set theory states that a particular element can also partially belong to a given set. The degree to which that element belongs to the set is expressed as a membership value or also referred to as its membership degree or membership grade. This membership value, defined on the interval [0,1], is determined by means of a membership function. When an element has a membership value of 1 it indicates that the element fully belongs to the particular set, a membership value of 0 indicates that the element is not part of that set. Any membership value between these extremes indicates a partial relation with the set (Klir & Folger 1988; Klir & Yuan 1995; Chang 1997; Burrough & McDonnell 1998; England 2000; Horstkotte 2001). In crisp set theory the transition of an element from belonging to a given set (membership value = 1) to not belonging to that set (membership value = 0) is described by means of an abrupt, step function (see Figure 2.3). In fuzzy set theory this transition is not necessarily a step function, but rather a gradual change from belonging to a set to not belonging to that set and is described by a membership function (see Figure 2.3).

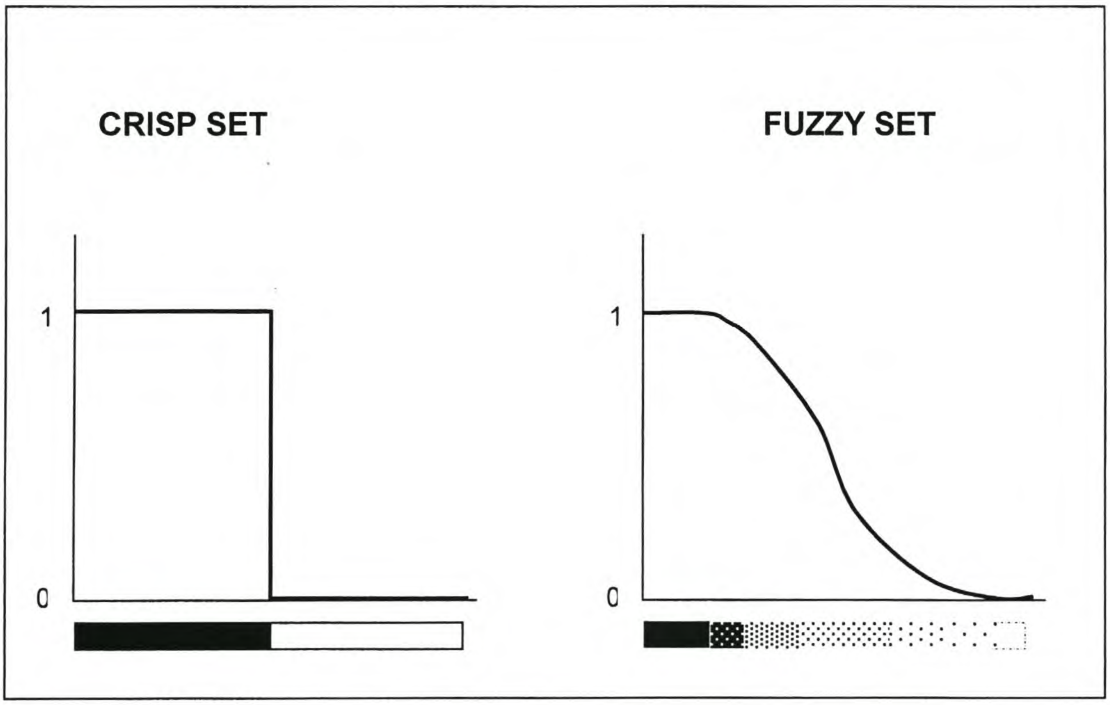


FIGURE 2.3 Crisp set vs. fuzzy set (adapted from Deist 1993)

Since full membership and full non-membership can still be indicated by the values of 1 and 0, respectively, the concept of a crisp set can be considered to be a restricted case of the more general concept of a fuzzy set for which only these two grades of membership are allowed. By assigning a step function as a membership function for a fuzzy set, the same result will be obtained as with normal crisp sets (Deist 1993). Fuzzy set theory should, according to Zadeh, not be interpreted as a single theory – the process of 'fuzzification' should rather be regarded as a methodology to generalise any specific theory from a crisp (discrete) to a continuous (fuzzy) form (England 2000).

The capability of fuzzy sets to express gradual transitions from membership to non-membership has a broad utility (Klir & Yuan 1995). It introduces vagueness (with the aim of reducing complexity) and so provides the tools to model real-world phenomena (Klir & Folger 1988). Unavoidable measurement uncertainties as well as vague concepts expressed in natural language which can best describe natural phenomena, can be incorporated into a model by using fuzzy sets (Deist 1993; Klir &Yuan 1995; Burrough & McDonnell 1998).

Fuzzy logic, which makes use of fuzzy sets, differs from classical logic in the sense that it can operate within that grey area between true and false (see Figure 2.3) and still come to a result. The use of fuzzy sets in fuzzy logic will now be discussed in the section that follows.

2.2.2 Fuzzy logic

Fuzzy logic is a superset of conventional (Boolean) logic that has been extended to support fuzzy sets so to be able to handle fuzzy concepts i.e. concepts of partial truth - truth values between 'completely true' and 'completely false'. It was introduced by Dr Lotfi Zadeh in the 1960s as a means to model the uncertainty of natural language (Zadeh 1992; Chang 1997; England 2000; Horstkotte 2001).

Fuzzy logic, like most other new technologies, developed slowly for the first 20 years and then the development accelerated rapidly when it was introduced in Japan, where it was applied to various products ranging from auto-focus cameras to subway control systems. Most of these early applications were in control systems (Chang 1997), which will be discussed in more detail in the next section.

Mathematical models of real world phenomena are based on functions that strive to map input variables to output variables. In many cases, however, a simple function cannot be used for the mapping, since the relationships between the variables are ill-defined, uncertain and complex. Much better results can often be obtained by using fuzzy logic, i.e. intuitive, understandable, linguistic terms to describe the phenomenon (Zadeh 1992; Klir & Yuan 1995; Chang 1997).

A fuzzy rule represents a fuzzy or approximate functional mapping from input variables to an output variable in a local region. Using a set of fuzzy rules, an approximate function between the input variables and the output variable for a region of interest can be specified (Dickerson & Kosko 1994; Chang 1997; Horstkotte 2001). An example of a fuzzy functional mapping between variable X and Y is depicted in Figure 2.4, where each circle or ellipse represents a fuzzy rule.

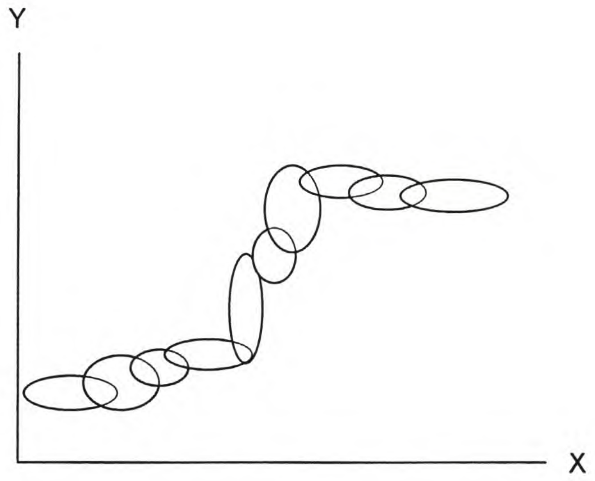


FIGURE 2.4 Fuzzy functional mapping using fuzzy rules (adapted from Chang 1997)

A very good approximation of the function can be obtained when many fuzzy rules with small local domains are used, but too many of them can increase computation time (Dickerson & Kosko 1994).

A fuzzy rule is an *if - then* statement that describes the output behaviour or the characteristics of a system being modelled as a ‘function’ of the inputs.

Usually more than one fuzzy rule is needed to describe a system. The collection of fuzzy rules describing a system is known as its fuzzy rule base or knowledge base. A fuzzy rule is stated in natural language. It is straightforward, intuitive and easy to understand (Klir & Yuan 1995; Chang 1997; England 2000; Horstkotte 2001).

The following is an example of a fuzzy rule:

1.00 **if** size **is** *large* **and** distance **is** *near* **then** risk **is** *moderate* (2.1)

Each rule consists of three basic parts (different fonts have been used and some words are in bold and underlined for explanatory purposes only – i.e. there are no strict rules to follow in this regard):

1.00	- the weight of the rule
if size **is** *large* **and** distance **is** *near*	- the antecedent (premise)
then risk **is** *moderate*	- the consequent (conclusion)

The weight of a fuzzy rule is a number in the interval [0,1] that defines the relative influence of the rule with respect to the other rules in the rule base. It is sometimes also referred to as the degree of certainty (DOS) of the rule (England 2000). The DOS must not be confused with the rule's degree of fulfilment (DOF), which will be discussed shortly. The DOF is calculated when evaluating the rule, whereas the DOS is an input parameter that is normally set to 1, unless there is reason to believe that a rule has less influence than the other rules in the rule base.

The antecedent part of a rule (also referred to as the premise or preposition) consists of the keyword **if**, followed by one or more fuzzy propositions combined with fuzzy operators. Fuzzy propositions and fuzzy operators will now be defined.

Fuzzy propositions are like normal sentences that can fundamentally be broken down into a subject and a predicate. In other words, a proposition can be expressed in general in the canonical form:

$$x \text{ is } P \tag{2.2}$$

where x stands for any subject from a designated universe of discourse X and the predicate P is a function defined on X and is denoted by P(x). The predicate, in fact, characterises a property of the subject x (Klir & Folger 1988).

Generally the subject x is referred to as input variables (the underlined words in 2.1) and the predicate P is referred to as a membership function (the italicised words in 2.1) defined on the domain of the input variables.

Fuzzy operators consists of the following:

is	- The membership grade evaluation operator
is not	- The complement of the membership grade operator
and	- The fuzzy conjunction operator
or	- The fuzzy disjunction operator.

The syntax of the **and** and **or** operators are as follows:

fuzzy proposition **and** fuzzy proposition
fuzzy proposition **or** fuzzy proposition

The **and** operator has a higher precedence than the **or** operator (as in Boolean logic). Parentheses can be used to override this precedence. Normally it is possible to choose between two possible definitions (evaluation modes) of the **and** and **or** operators, as indicated in Table 2.1.

TABEL 2.1 Evaluation modes of the 'and' and 'or' operators

Evaluation mode	**Operator**	
	X **and** Y	X **or** Y
MIN - MAX	Min (X,Y)	Max (X,Y)
Product & Probabilistic sum	X . Y	X + Y - X . Y

The consequent part of a rule (also referred to as the conclusion or result) consists of the keyword **then**, followed by one or more fuzzy propositions The propositions in this case consist of output variables and membership functions defined on the domain of these output variables. If there are more than one fuzzy proposition in the consequent part of a rule, then they are usually listed one after the other using commas as separators.

When the membership functions (for both the input and output variables) and the rule base have been defined, then the system should be ready to calculate an output value for a given input value by evaluating the rule base and using the membership functions. This part of fuzzy logic, where the rules in the rule base are being evaluated, is known as the inference process. The inference process consists of the following sub-processes: *fuzzification*, *evaluation* (sometimes also referred to as inference), *composition* and *defuzzification* (Berenji 1992; Klir & Yuan 1995; Chang 1997; Horstkotte 2001). The defuzzification sub-process is optional. The inference process and its sub-processes will now be discussed:

- In the *fuzzification* sub-process the membership functions defined on the input variables are applied to their actual values. In other words, the fuzzy operators (if any) are applied (according to their definition and precedence) to the fuzzy propositions in the premise part of a fuzzy rule. The result is multiplied by the weight (DOS) giving the degree of fulfilment (DOF) of the rule's premise (which is also referred to as a rule's degree of truth or its alpha). If a rule's premise has a non-zero DOF then the rule is said to fire.

- In the *evaluation (inference)* sub-process the DOF for the premise of each rule is applied to the conclusion part of each rule to calculate a modified output membership function for the rule that will define the output fuzzy set for the rule. There are two inference methods: *min* and *product* inferencing. In min inferencing, the output membership function is clipped off at a height corresponding to the computed DOF of the rule's premise. In product inferencing, the output membership function is scaled by the computed DOF of the rule's premise.

- In the *composition* sub-process all of the fuzzy sets assigned to each output variable are combined together to form a single fuzzy set for each output variable. This means that the output membership functions defined for each output variable of each rule are combined together into a single membership function representing the possible output values for that variable. There are two composition methods: *max* composition and *sum* composition. In max composition the combined output membership function is constructed by taking the point-wise maximum of all the membership functions assigned to the output variable during the previous evaluation (inference) sub-process. In sum composition the combined output membership function is constructed by taking the point-wise sum of all the membership functions assigned to the output variable during the previous evaluation (inference) sub-process.

- In the *defuzzification* sub-process, the output fuzzy set resulting from the previous composition sub-process, is converted to a single crisp value. There are numerous defuzzification methods, but the two most common are the *centroid* method (also sometimes referred to as the *centre of area* method, or the *centre of gravity* method) and the *maximum* method. In the centroid method the crisp output value is computed by finding the output value (x co-ordinate) of the centre of gravity of the output membership function obtained after the composition sub-process. In the maximum method the crisp output value is taken as the output value (x co-ordinate), where the output membership function (after composition) has the maximum degree of truth (at max y co-ordinate). There are several variations of the maximum method that differ only in what they do when there is more than one output variable at which this maximum truth value occurs. One of these, the *average-of-maxima* method, returns the average of the output values at which the maximum truth value occurs. In some applications only the output fuzzy set is of interest and defuzzification is then not applicable.

2.2.3 Fuzzy-logic-based controllers (fuzzy controllers)

Since 1985 fuzzy-logic-based controllers, also referred to as fuzzy controllers, have been used in more than 2000 industrial and consumer products such as light rail systems, washing machines, vacuum cleaners, camcorders, elevator control systems, air conditioners, automobile transmissions, anti-lock braking systems and TV enhancements (Chang 1997). There are also commercial applications of fuzzy logic in other areas such as speech and image recognition and decision support, although these applications are not as widespread as in control systems (Jamshidi 1994; Klir & Yuan 1995; Chang 1997).

In general a fuzzy controller is a special expert system (see Section 2.3.3.1) based on fuzzy set and fuzzy logic theory. It employs a knowledge base (also referred to as the fuzzy rule base) consisting of relevant fuzzy inference rules and an appropriate inference engine to solve a given control problem (see Figure 2.5). The fuzzy controller operates in a continuous closed cycle in which it receives (via an input port) an input value, usually a measured value that represents a relevant condition of the controlled process. The inference engine then evaluates the fuzzy rules that are stored in the knowledge base. The result of this evaluation is an output fuzzy set which is then converted to a crisp value via the defuzzification subsystem. The defuzzified output values represent actions taken by the fuzzy controller in the controlled process (Klir & Yuan 1995; Londong & Sauer 2001).

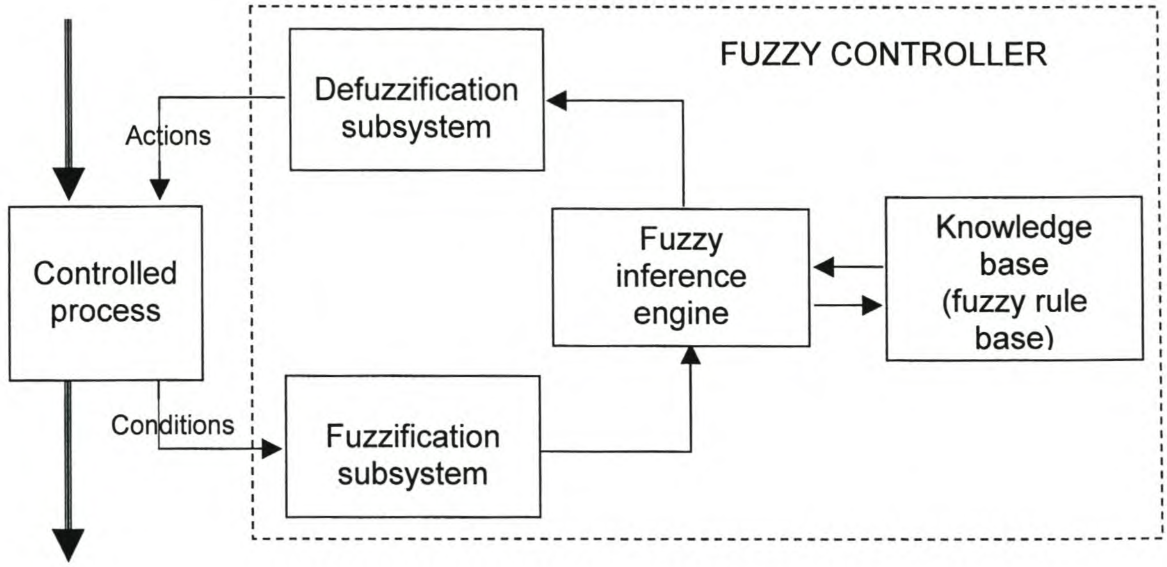


FIGURE 2.5 A general scheme of a fuzzy controller (adapted from Klir & Yuan 1995)

In applications other than controlled processes, such as data classification, which will be used in this dissertation for pipe break susceptibility analysis and impact assessment, the fuzzy controller functions in very much the same way. The output from the defuzzification system, however, is in this case not fed back into the 'controlled' system. In these types of applications independent input values are sent continuously to the fuzzification subsystem of the fuzzy controller for which final output values are to be calculated by the inference and defuzzification subsystems. The final output values will then simply be displayed or used in some other subsystem.

The knowledge base consists of a set of associations of the form 'If antecedent condition holds, then consequent condition holds'. These associations of fuzzy *if - then* rules relate input and output fuzzy sets. A fuzzy rule defines a fuzzy associative memory (FAM) transform from the input fuzzy set A to the output fuzzy set B. The knowledge base consists of a bank of FAM rules that stores expert knowledge or learned relationships between system variables (Dickerson & Kosko 1994; Jamshidi 1994; Klir & Yuan 1995; Londong & Sauer 2001). They are often linguistic descriptions of the control process and hereby employ human reasoning, which is often more appropriate to model the complex control processes with, than using mathematical functions (Klir & Folger 1988; Zadeh 1992; Klir & Yuan 1995; Londong & Sauer 2001).

If the rules in the rule base consist of two input variables and one output variable, then the rule base can be presented in a matrix form and the output value can be shown as a control surface (see Figures 7.7 and 7.14, which show the control surfaces obtained from the fuzzy controller of the SDSS when applied to the study area of the dissertation). The fuzzy controller produces the control surface by continuously plotting the resulting output values for automatically generated input values at regular intervals within the domain of the input variables. The control surface is a control measure for visually inspecting the rule base before starting the actual controlled process.

The membership functions used in the rule base are usually simple triangles and trapeziums, since these are computationally faster. It is found that many applications are not overly sensitive to variation in the shape of the membership functions - for which these simple membership functions can thus offer sufficient modelling capabilities (Klir & Yuan 1995). The parameters of the simple membership functions (triangles and trapeziums) can also be calibrated or fine-tuned more easily. This is known as machine training. In a pattern recognition type of problem (and also data classification applications) the parameters of the membership functions used in the rule base can be adjusted step-wise to minimise the error, which can be defined as the number of patterns in a training set that are incorrectly classified. When specifying memberships functions for the output variables i.e. consequent (conclusion) part of a fuzzy rule, it is good practice to make sure that their sizes are approximately equal so that none of the membership functions can dominate the others (Chang 1997). When categorising data into classes such as *small*, *medium* and *large,* then the singleton membership function (see Figure 2.6) is often the best choice to be assigned to the output variables.

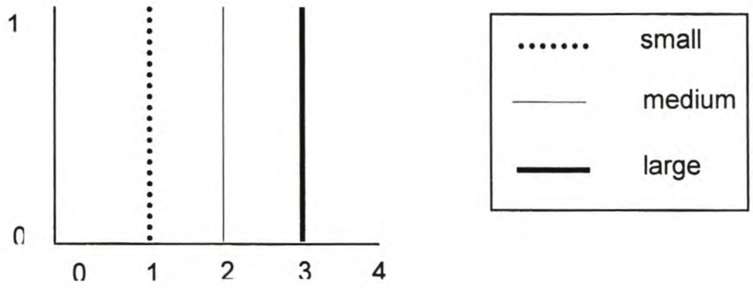


FIGURE 2.6 Singleton-type membership function

By using the singleton membership function for the output membership functions, the centroid defuzzification method (see Section 2.2.2) can be greatly simplified, because the area under these membership functions is then reduced to zero. The simplified centroid defuzzification method, also sometimes referred to as the *singleton* defuzzification method (Chang 1997), is based on equation 2.3, and calculates the output value as the weighted average of the centroids of the membership functions defined for the output. The weighting factors are the membership grades of the corresponding membership functions.

$$y = \frac{\sum_{i=1}^{n} a_i \cdot x_i}{\sum_{i=1}^{n} a_i} \tag{2.3}$$

y - output value

a_i - membership grade of membership function i

x_i - centroid of membership function i

n - number of membership functions defined for the output

Fuzzy controllers often provide the option to calculate the output values according to the singleton defuzzification method as described above, no matter what the shape of the output membership functions. It will thus neglect the area under an output membership function (as if it were a singleton output membership function) to simplify and speed up the calculations.

2.3 COMPUTER-BASED INFORMATION SYSTEMS

This section gives an overview of the different types of computer-based information systems. Information systems can also be non-computer-based, but these systems are not relevant to this dissertation and will not be discussed in this section. The exact classification of information systems, however, is difficult – maybe even impossible – since these systems have tended to become ever more integrated with each other, and thus the boundaries have blurred between categories that, not long ago, were quite distinct. There are three important stages in the evolution of information systems, viz. advancement from operational information systems to management information systems and then to decision support systems and executive information systems that together represent the final (present) evolutionary stage. The information systems in the earlier evolutionary stages, however, have not become obsolete, since they are still used today in many lower-level applications.

The following section will summarise the main characteristics of the information systems through the evolutionary stages.

2.3.1 Operational information systems

Operational information systems, also referred to as electronic data processing (EDP) systems, were used to automate the day-to-day record-keeping tasks in an organisation. The earliest commercial information systems were operational ones, because routine and repetitive tasks that involve little judgement in their execution are the easiest to automate. These first EDP systems were mainly focused on exploiting the computational speed and memory capacity of a centralised computer to perform repetitive computations on large data files, process transactions and create summary reports (Adam & Ebert 1986; Bennett, McRobb & Farmer 1999).

From a systems point of view, an operational information system (i.e. an EDP system) is located in the central part of the system depicted by the box labelled 'What the system does' in Figure 2.1 (Bennett, McRobb & Farmer 1999).

A lower-level manager (also referred to as a first-line manager) who makes short-term and well-structured types of decisions will typically benefit from using an operational information system (Adam & Ebert 1986; Grimshaw 1994).

2.3.2 Management information systems

A management information system (MIS) is an integrated, man/machine system for providing information to support the operations, management and decision-making functions in an organisation. The system utilises computer hardware and software, manual procedures, management and decision models, and a database (Davis 1974).

Duffy and Hough (1985) distinguish between a management support system (MSS) and a MIS. They argue that MSS differ from MIS only in that the former are computer-based whereas MIS are not necessarily so. In the literature this distinction is, however, not always strictly adhered to and the terms MIS and MSS are often treated as synonymous and interchangeable.

Information systems that are intended to support management usually work on a much higher level of complexity than operational systems. Much of the information used by management to make decisions is, however, derived directly from information stored at the operational level. In practice many management support systems (MSS) are therefore built on top of operational systems, in a hierarchical structure, to extract data from the operational systems and analyse or combine them to give managers information about the part of the organisation for which they are responsible (Emery 1969; Bennett, McRobb & Farmer 1999). A MSS is designed and structured to provide pre-defined flows of information, using an integrated database and specific-purpose models for selected problem areas. The models incorporated in a MSS are separate stand-alone models. The user must therefore search the system for the appropriate model applicable to his problem field and then formulate the queries that are supported by that specific model (Adam & Ebert 1986).

From a systems point of view, a MSS is located in the control unit of the system depicted by the box labelled 'How the system is controlled' in Figure 2.1. Thus the crucial aspect of a MSS is the feedback or the feed-forward that it provides, alerting managers to problems and opportunities, and assisting them in the process of fine-tuning the organisation's performance (Bennett, McRobb & Farmer 1999).

A MSS is generally aimed at the middle manager for supporting both semi-structured and structured decision-making (Duffy & Hough 1985; Grimshaw 1994).

2.3.3 Decision support systems

A decision support system (DSS) is a computer-based system that helps the decision-maker utilise data and models to solve unstructured problems (Sprague & Carlson 1982). These unstructured problems can be either very complex and difficult to solve, or they are loosely defined, so that the decision-maker's definition of the problem and his identification of key factors are continually revised during the investigation (Abel *et al.* 1992; Shen & Grivas 1996; Srinivasan, Sundaram & Davis 2000).

DSS are decision-focused and, accordingly, they emphasise flexibility, adaptability and quick response, i.e. they are user-initiated and user-controlled (Sprague & Carlson 1982). A DSS provides an ad-hoc problem solving environment and supports 'what-if' analyses. Managers can explore the solution space, i.e. try out different decisions, alter input data, and quickly see the results of these changes in the solutions to problems (Densham 1992; Render & Stair 1994; Srinivasan, Sundaram & Davis 2000). The DSS was developed as a response to overcome the shortcomings of the MIS regarding analytical modelling capabilities and the decision maker's interaction with the solution process (Densham 1992). A DSS uses a confederation of models that are user-activated in diverse combinations. The output of one model can be the input of another model, so that complex modelling combinations can be specified. The models can, however, also be used as stand-alone models as in a MSS (Adam & Ebert 1986). A DSS can provide a choice of the presentation form (Duffy & Hough 1985) so that different views of the final model results can be obtained.

From a systems point of view a DSS, just as the MSS (see Section 2.3.2), is located in the control unit of the system (see Figure 2.1). A DSS, however, offers more decision-making support than a MSS, and will therefore also provide more extensive feedback or feed-forward than a MSS.

In terms of taxonomy a DSS is aimed at senior managers to support their semi-structured as well as unstructured, strategic decision-making. The senior managers will still make the final decisions, since the DSS is only a supportive tool and will by no means replace managerial judgement (Keen & Scott-Morton 1978; Duffy & Hough 1985; Grimshaw 1994; Render & Stair 1994; Malczewski 1997).

According to the Scott-Morton classification scheme, a DSS is a specialised MSS system. An *expert system,* according to the above-mentioned classification scheme, is a special type of DSS and can be incorporated into a standard DSS, especially when the type of problem is very complex (Duffy & Hough 1985). A *spatial decision support system (SDSS)* is also a special type of DSS that extends the standard DSS in order to support spatial analysis (Densham 1992). Expert systems and spatial decision supports systems (SDSS) will now be discussed.

2.3.3.1 Expert systems

An expert system models the reasoning process of a human expert within a specific domain of knowledge in order to make the experience, understanding and problem-solving capabilities of the expert available to the non-expert for purposes of consultation, diagnosis, learning, decision support or research (Klir & Folger 1988). An expert system includes all equipment, devices, procedures and software used to capture the essence of a human expert in a particular field. Expert systems are also a primary application in the field of artificial intelligence (Duffy & Hough 1985; Render & Stair 1994).

An expert system is built by one or more individuals called knowledge engineers. Knowledge engineers have the expertise and skills to convert information given to them by experts in a field into computerised systems that act as experts. There are numerous advantages to capturing expertise. For instance, the knowledge and experience of the expert are permanently stored in the system and can be used for generations to come. In addition, the knowledge and experience can now also be used by many other individuals in a variety of settings to solve their own specific problems (Klir & Folger 1988; Duffy & Hough 1985; Render & Stair 1994).

An expert system consists of three basic components, viz. a *user interface*, a *knowledge base* and an *inference processor (or inference engine)*. The interface provides user-friendly access to the expert system. The data and experience that are captured are placed into the knowledge base of the expert system. A *rule base*, which is sometimes regarded as a forth component of the expert system but more often as part of the knowledge base, contains important rules about the particular field that is captured in the expert system. The rules in the rule base consist of *if - then* constructs and are often referred to as production rules. Finally, the expert system has an inference engine that allows the manipulation of the rule base and knowledge base to get meaningful results for the user (Klir & Folger 1988; Duffy & Hough 1985; Render & Stair 1994).

The facts, relations, judgements, opinions and 'rules of thumb' contained within the knowledge base usually manifest varying degrees of imprecision and uncertainty. It is nevertheless desirable for an expert system to be able, like the human expert, to draw non-trivial inferences from imprecise data and vague heuristics (Klir & Folger 1988). The rule base of the expert system should therefore be able to support fuzzy rules and the inference engine should be able to perform fuzzy inference (see Section 2.2.2) on the fuzzy rules. This can be established by linking a fuzzy controller (see Section 2.2.3) via special interfaces to the expert system.

A DSS consisting of an expert system that is linked to a fuzzy controller can further be extended to support spatial analysis as well. This will be discussed in the following section.

2.3.3.2 Spatial decision support systems

A SDSS is an extension to the standard DSS to assist decision-makers with complex spatial problems. A SDSS basically provides a framework for integrating database management systems with analytical models, graphical display and tabular reporting capabilities, and the expert knowledge of decision-makers. A problem-solving environment can thus be established in which the decision-maker can explore, structure and solve complex spatial problems (Densham 1992; Malczewski 1997; Frank, Thill & Batta 2000).

Decision-makers have increasingly been turning to geographical information systems (GIS) to assist them with solving complex spatial problems (Densham 1992; Grimshaw 1994; Arentze & Timmermans 2000). The GISs of the 1990s were, however, not yet suitable to support complex decision-making. Densham (1992) and also Arentze and Timmermans (2000) claim that a GIS lacks sufficient analytical modelling capabilities to be able to support complex decision-making. Information exchange between GIS and the user occurs mainly via digital maps and the database. A decision-maker who is exploring a problem as well as generating and evaluating alternative solutions will also require other forms of information exchange such as graphs and reports. The graphical and tabular reporting capabilities of a GIS are often not sufficient for the decision-making process.

Arentze & Timmermans (2000) developed a SDSS for retail plan generation and impact assessment, in which complex analytical modelling was carried out by a special module, which is loosely linked to a GIS (via data files) for the spatial representation of the results. Frank, Thill and Batta (2000) developed a SDSS for hazardous material truck routing. They considered a link to a GIS for data input and graphical display of the results, however, had difficulty with the data transfer speed.

There have been major developments in GIS in the recent years. Modern GIS software has powerful built-in object-oriented programming languages (ESRI 2001), which now makes it possible to add analytical modelling capabilities to a GIS. With the built-in programming languages interfaces can also be written to provide seamless links to other software packages such as spreadsheets, for adding further tabular and graphing functionality to the system, and seamless links to fuzzy controllers for enabling fuzzy interference. The built-in object-oriented programming languages of modern GIS are very expressive, so that most of the analytical modelling can be incorporated entirely within the GIS and very few links need to be established with third-party external software products (the linkage to external products can, however, largely reduce system development time). A modern GIS can therefore be a central component around which a successful SDSS (with an expert system also incorporated) can be developed.

The design of a SDSS should be based on the DDM (dialog, data and model) paradigm. According to this paradigm, a well-designed SDSS should have a balance among the three DDM components and should consist of the following: a Data Base Management System (DBMS) that contains the functions to manage the geographic data base; a Model Base Management System (MBMS) that contains the functions to manage the model base; and lastly, the SDSS should have a Dialog Generation and Management System (DGMS) that manages the interface between the user and the rest of the system. A GIS can play a vital role in the first component of DDM, viz. to manage and display the geographic data needed for the spatial decision-making (Malczewski 1997). The design and development of information systems (which are also applicable to SDSS design and development) will be discussed in detail in Section 2.4.

2.3.4 Executive information systems

An executive information system (EIS) provides easy-to-use functions to evaluate summarised data from various sources. The EIS integrates data from different system components and from different data sources both within and outside the company. This enables a quick overview of all critical success factors of a company at any time (SAP 1997).

In terms of taxonomy the EIS is aimed at the most senior managers, viz. the executives. An EIS supports essentially strategic and unstructured decision-making regarding the overall company policy (Grimshaw 1994). The information source (input) of an EIS is usually the output from some other system such as a DSS or a SDSS. Detailed technical analyses are normally not performed with an EIS, since the type of information reaching the executive level is already in a summarised form.

2.4 INFORMATION SYSTEMS DEVELOPMENT METHODOLOGY

Information systems development methodology has evolved over the years from the *pre-methodology era*, to the *methodology era*, and finally to the present *post-methodology era*. Avison and Fitzgerald (1999) describe the methodology eras as follows:

In the *pre-methodology era* the focus was on computer programming and the solution of various technical hardware issues. This emphasis meant that the requirements of the users and of the organisation were somewhat neglected. This was originally not such a great problem as the systems were usually relatively straightforward, with requirements well defined. But as the complexity of the systems being developed increased, it became more of a problem. The needs of the users were rarely clearly established, with the consequence that the system designs were often inappropriate for the real requirements of the application and the business.

The *methodology era* introduced a recommended series of steps and procedures to be followed in the course of developing an information system. This gave more structure to the development process. The Structured Systems Analysis and Design Method (SSADM) is a well-known and successful method from this 'structured' software development era, and was for many years a UK government standard for system development and procurement. These step-by-step prescriptive development methodologies, also referred to as structured methodologies, however, have been criticised for not being flexible enough to be used in the dynamic world of information systems development. As information systems requirements have become increasingly complex, the use of an object-oriented approach has become more necessary. The latest object-oriented systems development methodology, with its iterative style, suits the natural planning process in information systems development. The Rational Unified Process is a very successful object-oriented systems development methodology. It supports object-oriented techniques and is a configurable process to fit the needs of most system development projects (Booch, Rumbaugh & Jacobson 1999).

The current *post-methodology era* is characterised by a large number of different types of methodologies rather than the development of one 'common' methodology. It is believed that systems development will become more contingent, where a framework is presented but tools and techniques are expected to be used or not (or used and adapted), depending on the situation. Object-oriented modelling techniques supported by the use of Computer-Aided Software Engineering (CASE) tools are believed to play an important role in the development methodologies of this era. CASE tools offer automated diagramming support that will ensure that all phases in the development of information systems are properly documented. The Unified Modelling Language (UML) is expected to become the standard documentation language (see Section 2.4.2) and is supported by most CASE tools. The automatic code-generation capability of the CASE tools will further be improved so that information systems can be implemented more rapidly.

An information system development methodology basically consists of a life-cycle model to structure the development process, an approach to software development, e.g. object-orientation (see Section 3.1), and a set of techniques and notations to support and document the approach (Bennett, McRobb & Farmer 1999).

2.4.1 Development life-cycle

The subdivision of software development processes is known as life-cycle models. The earliest life-cycle models are the traditional waterfall life-cycle models, then followed the spiral models for incremental development and the latest are the object-oriented life-cycles. The various development processes are subdivided into different phases and after completion of the phases a complete and appropriate system can be established. The life-cycle models all have in common the following basic phases: *analysis*, *design*, *implementation*, *testing*, *delivery* and *maintenance*. The implementation phase is often also referred to as *construction* and the delivery phase is also known as *deployment or installation*. The sequence of these phases can differ. In the waterfall models the phases follow a linear (i.e. sequential) execution (with possible iteration allowed in some of the phases of the more sophisticated waterfall models); the spiral model will usually always have iteration in its phases; and the object-oriented life-cycle models have phases that can run parallel, i.e. all taking place at the same time (Caspers 1994; Kösters, Pagel & Six 1997; Bennett, McRobb & Farmer 1999; Booch, Rumbaugh & Jacobson 1999).

The focus will mainly be on the analysis, design and testing phases when developing the SDSS of this dissertation. The analysis and design phases can be seen as the system blueprints and are of utmost importance to the successful delivery of the final system. The SDSS to be developed in this dissertation will finally be tested by applying it to a certain study area (see Chapter 7). The delivery and maintenance phases, however, are not within the scope of this dissertation.

In the earlier structured approach to system development, a clear distinction between analysis and design was made in terms of the types of diagram that were used to model these development phases. During analysis data flow diagrams were used to model requirements, whereas structure charts or structure diagrams were used to model the design of the system and the program it was based on. In object-oriented systems development methodology, however, there is not always a clear distinction made between analysis and design, i.e. the two phases are seamless since the same object model is used right through the life of the project (Bennett, McRobb & Farmer 1999). Rumbaugh (1997) distinguishes between analysis and design in terms of the amount of detail that is included in the model. On a continuum the analysis phase provides an abstract model of 'what the system should be able to do', while the design phase documents 'exactly how to do it'. As the project progresses from one end of this continuum to the other, additional details are added to the model until a clear specification of 'how to do it' is provided.
The analysis and design phases of information system development will now be discussed.

2.4.1.1 Analysis

The analysis phase in information system development is more generally known as requirements analysis. Other terms that are also often used are requirements capture, requirements specification or requirements engineering. It is basically a method to determine and document precisely what the users expect the system to do for them. The overall objectives of the company or organisation must be well understood, as must the individual user's expectations of the system. Requirements analysis thus defines the problem domain and elaborates a model (a specification), which defines the system's goals, data, functionality and constraints. Without a clear definition of the requirements of the system it will be difficult, if not impossible, to design and build databases, select commercial products and finally deliver a system that can be used properly in the organisation. Wrong, missing or ambiguous requirements that are first discovered at the final development stages (viz. system testing and delivery) are extremely costly to correct, since the necessary changes to be undertaken will then affect almost all previous development stages (Huxhold & Levinsohn 1995; Kösters, Pagel & Six 1997; Bennett, McRobb & Farmer 1999).

During requirements analysis existing information on all aspects of the environment or the phenomena that the system must address should be gathered. Existing systems in the organisation, if there are any, should also be studied, since some of the existing routines may be useful and can be incorporated into the newly improved system. All physical or organisational processes to be analysed should be broken down into the individual functions. This reduction may appear to be in opposition to the systems theory of looking holistically (see Section 2.1.3), but is not, provided that it is taken into account that a single component may function differently when isolated from the system. A complex function can be regarded as a subsystem with inputs and outputs (see Figure 2.2). For each function to be computerised, the user(s) must be identified, the function input and output flow (i.e. data source and output destinations) as well as possible constraints must be captured (Huxhold & Levinsohn 1995).

Information system users may often find it difficult to imagine how their requirements will be translated into a working system. Potential misunderstandings and ambiguities that may exist in the requirements specification can be overcome by prototyping the system. In information system development a prototype is a system or partially complete system that is built rapidly to explore some aspect of the system requirements and is not intended as the final working system. When the prototype is generally accepted, then the final system designs can be completed and the implementation can start (Caspers 1994; Kösters, Pagel & Six 1997; Bennett, McRobb & Farmer 1999).

To summarise, the requirements analysis focuses on *what* functions the system must provide and not *how* to construct these functions. How to construct the functions is part of the design phase, which will be discussed in the next section.

2.4.1.2 Design

Design has been described by Rumbaugh (1997) as stating *how* the system will be constructed without actually building it. Design is about producing a solution that meets the requirements that have been specified during the requirements analysis. The design activity is concerned with specifying how the new system will meet those requirements and the models that are produced during design will thus show how the various parts of the system will work together (Bennett, McRobb & Farmer 1999).

Design takes place at two main levels: *system design* and *detailed design*. *System design* is concerned with the overall architecture of the system. The system will typically be divided into subsystems that are allocated to different processes. The subsystems of information systems are often arranged according to the classical three-tier architecture, which consists of a User Services subsystem, a Business Services subsystem and a Data Services subsystem. The User Services subsystem provides the user interface for presenting information and gathering data. The Data Services subsystem maintains, accesses and updates the data. The Business Services subsystem bridges the gap between the two subsystems and manages requests from the user to execute a business task. The Business Services subsystem also includes business rules that dictate the policies for manipulating the data. The DDM paradigm (discussed in Section 2.3.3.2) for designing a SDSS is fundamentally based on this classical three-tier architecture of information systems design. *Detailed design* is concerned with the detailed working of the system and its subsystems. During detailed design additional layers of detail are added to the overall design model of the system until the internal working of all subsystems is fully described (Bennett, McRobb & Farmer 1999; Booch, Rumbaugh & Jacobson 1999).

2.4.2 Documentation

The functional requirements and design models must be properly documented, preferably according to some documentation standard. Standards are important as they promote communication in the same way as a common language. They enable communication between members of the development team and promote communication over time, even after several years, when system maintenance and modifications are to be done (Bennett, McRobb & Farmer 1999).

The Unified Modelling Language (UML), which is an expressive graphical language (see later Section 3.2) for representing the concepts in the development of an object-oriented information system, is expected to develop into an industry standard (Bennett, McRobb & Farmer 1999). UML version 1.1 was already adopted by the Object Management Group (OMG) in 1997 as a standard language for object-oriented modelling. Maintenance of the UML was taken over by the OMG Revision Task Force (RTF) to further refine the language and new versions of UML have come out since then (Booch, Rumbaugh & Jacobson 1999).

UML will also be used in this dissertation to document the requirements analysis and design models when developing the SDSS for pipe break susceptibility analysis and impact assessment.

2.5 SUMMARY

This chapter has provided an overview of information systems. The concepts of a system, the different types of computer-based information systems and information systems development methodologies have been discussed. The fuzzy-logic-based controller, a special type of computer-based information system, has been described in detail in this chapter, since it will be linked with the SDSS to conduct complex analyses. The next chapter will provide fundamental background on object-oriented modelling and will also describe the Unified Modelling Language (UML) diagramming techniques.

CHAPTER 3: OBJECT-ORIENTED MODELLING AND DIAGRAMMING SUPPORT

An object-oriented approach will be followed in this dissertation in developing the SDSS for pipe break susceptibility analysis and impact assessment. This chapter gives an overview of the basic concepts of object orientation, on which the pipe break susceptibility analysis models and impact assessment models of the SDSS will be based. The UML diagramming techniques that will be used to document these models will also be discussed.

3.1 BASIC CONCEPTS OF OBJECT ORIENTATION

In object-oriented modelling the basic building block is the object. Objects are often related to other objects and objects can interact with one another. These concepts will be discussed in the sections that follow.

3.1.1 Object, instance and class

An object is an abstraction of something in a problem domain. Objects serve two purposes: they promote understanding of the real world and they provide a practical basis for computer implementation. An object is an entity with a well-defined boundary and identity that also encapsulates state and behaviour. Where 'state' represents the particular condition that an object is in at a given moment, 'behaviour' stands for the things that the object can do, and 'identity' means that every object is unique and will get a unique object identifier assigned to it (Eaglestone & Ridley 1998; Bennett, McRobb & Farmer 1999; Booch, Rumbaugh & Jacobson 1999). The encapsulation principle will further be discussed in Section 3.1.5.

A popular definition of an object is given by Booch, Rumbaugh and Jacobson (1999) which clearly indicates the connection between the three concepts of object, instance and class, viz.: 'An object represents a particular instance of a class.' Objects that are sufficiently similar to each other are said to belong to the same class. Instance is another word for a single object, but it also carries the connotation of the class it belongs to. A class is an abstract descriptor for a set of instances with certain logical similarities to each other (Bennett, McRobb & Farmer 1999).

3.1.2 Attributes

Attributes are part of the essential description of a class. They describe the common structure of 'what an instance of the class can know'. Each instance has its own values for the attributes. When the attribute values change, the state of the object will also change (Caspers 1994; Eaglestone & Ridley 1998; Bennett, McRobb & Farmer 1999; Booch, Rumbaugh & Jacobson 1999).

3.1.3 Operations

Operations are the elements of common behaviour shared by all instances of a class. They describe the common structure of 'what an instance of a class can do'. They are thus actions that can be carried out by the object. Operations are also services that objects may be asked to perform by other objects. Each operation has a signature which is a definition of the interface through which the operational services can be accessed (see Section 3.1.5). Operations are eventually implemented by methods defined on the objects. An object can also only change its state through the execution of an operation, i.e. the attributes of an object cannot store or update their own values, and also links (see Section 3.1.4.3) cannot make or break themselves (Caspers 1994; Eaglestone & Ridley 1998; Bennett, McRobb & Farmer 1999; Booch, Rumbaugh & Jacobson 1999).

3.1.4 Relationships

In object-oriented modelling there are basically three kinds of relationships objects can have with one another: dependency, generalisation and association (Bennett, McRobb & Farmer 1999; Booch, Rumbaugh & Jacobson 1999).

3.1.4.1 Dependency

Dependency is a 'using' relationship to indicate that a change in state of the objects in one class (the independent class) can affect the objects in another class (the dependent class that 'uses' the independent class). Most often dependencies are used in the context of classes to show that one class uses another class as an argument in the signature of an operation.

3.1.4.2 Generalisation

Generalisations connect generalised classes to more specialised ones in what is known as sub-class/super-class or child/parent relationships. Generalisation is sometimes called an 'is-a-kind-of' or also 'is-a-type-of' relationship. The specialised (child) class is-a-kind-of the more general (parent) class. The child class inherits the properties (attributes and operations) of its parent(s). A class that has exactly one parent is said to use single inheritance, whereas a class with more than one parent is said to use multiple inheritance. A child class can have attributes and operations in addition to those inherited by the parent. An operation in a child class that has the same signature as an operation in its parent class overrides the operation in the parent class; this is known as polymorphism and will be discussed further in Section 3.1.6.

3.1.4.3 Association

Associations are structural relationships among instances. The concepts of link and association, like those of object and class, are very closely related. A link is a logical connection between two or more objects. Just as a link connects two instances, an association connects two classes, and just as a class describes a set of similar instances, an association describes a set of similar links (links are called 'association instances' by some authors). An association is a connection between two classes that represents the possibility of their instances participating in a link. This greatly simplifies the modelling process, since the modelling of every link separately would be uneconomical and complex, because there may be thousands of instances in a class participating in links between instances of other classes.

Linked instances may be from different classes or from the same class. Unary associations involve links between objects within the same class. An association that connects exactly two classes, which is most often the case, is called a binary association. Associations that connect more than two classes are called n-ary associations.

It is often important to define a limit on the number of links with objects of another specific class in which a single object can participate. Multiplicity (also referred to as cardinality) of an association defines upper and lower limits on the number of other instances to which any one object may be linked. Associations with the following generic multiplicity are often found: *one*-to-*one*, *one*-to-*many*, and *many*-to-*many*. The 'many' part, can have many different forms of constraints (also referred to as relationship rules) assigned to it to refine the generic multiplicity (cardinality) to specific values, which will be discussed in more detail in Section 3.2.2. Association multiplicity conveys important information about the structure of the problem domain. Different assumptions about multiplicity have significant effects on system design. If association multiplicities are modelled incorrectly, the system will then not function the way the user wants it to.

A plain association between two classes represents a structural relationship between peers, meaning that both classes are conceptually at the same level, no one more important than the other. Sometimes, however, it is necessary to model a 'whole/part' relationship. There are two kinds of 'whole/part' relationships, the 'has-a' relationship (also referred to as the 'contains' relationship) and the 'is-composed-of' relationship. *Aggregation* and *composition* are special types of associations that can be used to model the 'has-a' relationship and 'is-composed-of' relationship respectively. Aggregation states that an object of the whole has objects of the part, but is not composed of the parts. Whereas, composition models the relationship in which the whole is made up of (is composed of) the parts.

3.1.5 Object encapsulation and object interaction

Encapsulation means that an object contains both a state and an operation(s) that can be performed on that state (Sodhi & Sodhi 1999). An object can be considered as a unit (capsule) of data together with processes (operations) that act on the data (Microsoft 1997). This bundling of procedures and data is the foundation of the object-oriented concept (Bennett, McRobb & Farmer 1999; Sodhi & Sodhi 1999). Encapsulation is a way of protecting the data, since the encapsulated data are not freely accessible from outside the object (see Figure 3.1). Only an object's own operations (methods) can directly manipulate its data. Methods are invoked either by another method of the same object or by a call (message) from another object within the application program. Encapsulation hides the implementation details so that the user (or calling object) does not need to know how an object performs its function; the user only has to know what the object can and cannot do (Eaglestone & Ridley 1998; Bennett, McRobb & Farmer 1999; Sodhi & Sodhi 1999). This is analogous to the black box approach of systems theory as discussed earlier in Section 2.1.3.2. Also, when an object is added to the program code, unpredictable actions cannot occur to other parts of the program, since an object can only modify its own data (Sodhi & Sodhi 1999).

In practice it is not usually possible for all processes to be located together with the data that they must access, and data and processes are distributed among many different objects. Thus each individual operation within an object may represent only a small part of a larger task that is distributed among many objects. Object interaction, also referred to as message passing and collaboration, is the way that these distributed objects communicate with each other in order to accomplish the overall task. According to the encapsulation principle, an object knows only its own data and its own operations. But in order for collaboration to be possible, objects also need to know how to request services from other objects. The object should therefore know which other object to ask and how to formulate the question. An object's operations can thus only be called by a message with a valid operation signature (see Figure 3.1). Other objects can only access the data if they know the precise signature of the operations that access and manipulate the object data (see Figure 3.1). If the signatures to an object's operations are not changed, it makes no difference what changes are made to the way that the operations run or the way that the data are stored. Neither of these changes would be visible from the outside. Encapsulation not only protects the object's data, but it also hides the internal details of the object from other parts of the system, thus minimising the knock-on effects of any changes to the design or implementation (Caspers 1994; Eaglestone & Ridley 1998; Bennett, McRobb & Farmer 1999; Booch, Rumbaugh & Jacobson 1999; Sodhi & Sodhi 1999).

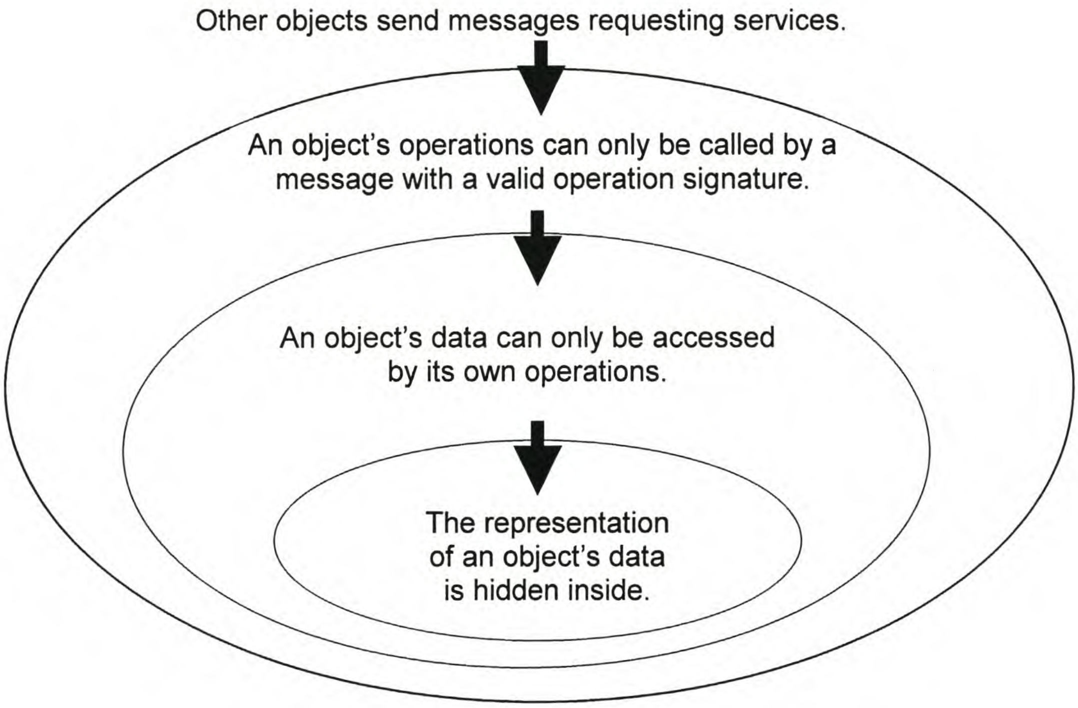

Other objects send messages requesting services.
An object's operations can only be called by a message with a valid operation signature.
An object's data can only be accessed by its own operations.
The representation of an object's data is hidden inside.

FIGURE 3.1 Encapsulation: The layers of protection that surround an object (adapted from Bennett, McRobb & Farmer 1999).

The operations of an object that can be called by other objects should be defined as public operations. Operations that can only be accessed internally within the object should be defined as private operations. Ideally the object data (viz. the object attributes) are always defined as private (as depicted in Figure 3.1). In practice, however, it is sometimes unavoidable that certain attributes should be publicly accessible (i.e. readable and writable from all classes). This accessibility of attributes and operations is known as object visibility and is basically a definition of the encapsulation boundary of an object (Bennett, McRobb & Farmer 1999).

In the following section polymorphism will be discussed; this is a useful concept that allows very flexible message passing between objects of different classes.

3.1.6 Polymorphism

Polymorphism refers to the possibility of identical messages being sent to objects of different classes, each of which responds to the message in a different yet still appropriate way. The key to this is that a receiving object is responsible for knowing how to respond to messages (Microsoft 1997; Bennett, McRobb & Farmer 1999; Sodhi & Sodhi 1999).

This ability to generalise behaviour over many types of objects allows a higher degree of abstraction in systems development, because the developer thinks of specific actions rather than the details of how they will be implemented (Sodhi & Sodhi 1999).

3.1.6.1 Polymorphism implemented by inheritance

Most object-oriented programming languages provide polymorphism through inheritance (Microsoft 1997). This means a sub-class can override the implementation of a particular method in order to implement a more specific operation which will be more appropriate or efficient than the one inherited from its super-class (Microsoft 1997; Bennett, McRobb & Farmer 1999; Booch, Rumbaugh & Jacobson 1999; Sodhi & Sodhi 1999).

3.1.6.2 Polymorphism implemented by late binding

Late binding occurs when the compiler cannot determine at compile time what kind of object a variable will contain. In the example shown in Figure 3.2 the *Shape* argument is declared as *Object*, so at run-time it can contain a reference to any kind of object. In order to check if the methods invoked on that object, viz. *area* and *perimeter,* are valid (see Figure 3.2), the compiler compiles some extra code. This extra code will at run-time check if the object supports the method and if so, then it will be invoked, otherwise an error message will be raised. Late binding, also referred to as dynamic binding, is thus a mechanism for determining at run-time which method belongs to which object (Caspers 1994). The operation shown in Figure 3.2 can be implemented as a method of a 'calc_Shapes' class, for example, which can then receive all sorts of different shapes for which the area and perimeter can be calculated.

```
Public Sub calc_Surface(ByVal Shape as Object)
Shape.Area
Shape.Perimeter
End Sub
```

FIGURE 3.2 Polymorphism implemented by late binding.

Some compilers support the use of interfaces which will allow the compiler to check the type library at compile time to see if that interface supports the method that is being called. This is referred to as early binding and will give better speed performance at run-time than late binding (Microsoft 1997).

3.1.7 Collection objects

A collection object is an ordered set of items that can be referred to as a unit. The collection object thus provides a convenient way to refer to a related group of items as a single object. The items, or members, in a collection need only be related by the fact that they exist in the collection. Members of a collection can be any of the standard data types (such as integer, string, double, etc.), as well as user-defined data types and objects derived from specific classes. The latter implies that a collection can thus contain other complex objects that, for example, also have collection objects as attributes.

The members of a collection also do not have to share the same data type, which makes it a very flexible structuring mechanism (Microsoft 1997).

Once a collection is created, members can be added using the *add* method and removed using the *remove* method. Specific members can be returned from the collection using the *item* method, while the entire collection can be iterated using the *For Each...Next* statement (Microsoft 1997). The above syntax is that of Visual Basic, but will be very similar in other programming languages.

3.2 UNIFIED MODELLING LANGUAGE DIAGRAMS

The Unified Modelling Language (UML) is a graphical language for visualising, specifying, constructing and documenting the artefacts of a software-intensive system. UML is not an information system development methodology and is largely process-independent, meaning that it is not tied to any particular software development life-cycle (see Section 2.4.1). It basically provides a standard way of writing the blueprints of a system, and hence promotes better communication (see Section 2.4.2). The UML graphical language includes nine diagrams for describing different aspects and architectural views of a system. An architectural view of a system according to UML is a projection into the organisation and structure of the system at a particular stage in the development. The UML architectural views of a system consist of five interlocking views: the use case view (which is basically a view of the system requirements, see Section 2.4.1.1), the design view, the process view (very similar to the design view, but the focus is on the system's concurrency and synchronisation mechanisms), the implementation view and the deployment view (Booch, Rumbaugh & Jacobson 1999).

The UML Package (see Section 3.2.6), which will be used in this dissertation as a mechanism for organising subsystems, will also be discussed in this section, although it is not formally one of the nine UML diagrams. The specific UML diagrams that will be deployed in this dissertation will be discussed in the sections that follow.

3.2.1 Use case diagram

A use case diagram helps to visualise and capture the intended behaviour of the system to be developed. It specifies the functionality which the system will offer from the user's perspective. The use case diagram is mainly used in the requirements analysis phase of system development (see Section 2.4.1.1) and can document the requirements of the whole system as well as the requirements of individual subsystems.

The use case diagram provides a description of the interaction between the users of the system, termed *actors*, and the high-level functions within the system, represented graphically by the oval use cases (see Figure 3.3). The actors need not only be human users of the system; they can also be other automated systems that communicate with the system. Communication lines on the use case diagram show the communication between actors and use cases. Two types of relationships between use cases are depicted on the diagram. These are the *uses* relationship, and the *extends* relationship. The *uses* relationship, also referred to as the *include* relationship, is applied when there is a sequence of behaviour that is used frequently in a number of use cases. To avoid copying the same description in each use case, the commonly used sequence of behaviour is put in a separate use case to which a link can be made when needed. The *extends* relationship can be applied to model optional system behaviour (e.g. when the user chooses the flow of events at menu options) and also other variations in system behaviour invoked by the system itself (e.g. at if-statements). Core functionality is thus placed in one use case, which is then extended via extension points to other use cases that optionally do something that is based on the core functionality.

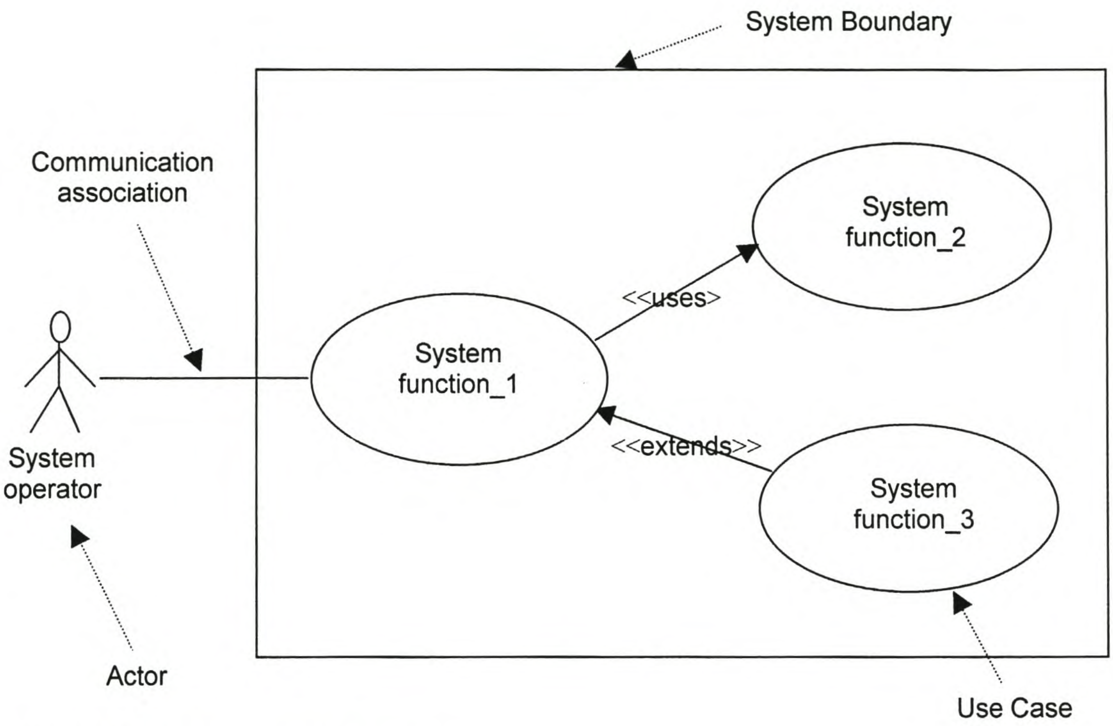


FIGURE 3.3 Use case diagram

The use case diagram is also accompanied by textual use case descriptions (usually in tabular from) that describe step by step the actor action and the corresponding system response of each use case in the diagram.

3.2.2 Class diagram

Each class in a class diagram is shown by a rectangular box. The rectangular class boxes are subdivided into three compartments (see Figure 3.4) to distinguish between class name, attributes and operations.

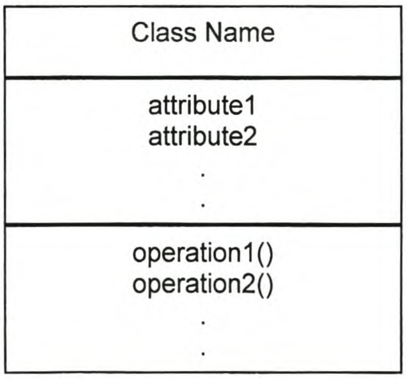


FIGURE 3.4 Class diagram - compartments

The visibility (i.e. accessibility) of the attributes and operations in a class (see Section 3.1.5) is indicated on the class diagram by preceding the attribute or operation names with the symbols shown in Table 3.1.

TABLE 3.1 Visibility of class attributes and operations

Visibility Symbol	Visibility	Meaning
+	Public	Accessible by any class.
-	Private	Accessible only by the class that includes it.
#	Protected	Accessible only by the class that includes it or a sub-class of that class.

The UML diagramming notation used in the class diagram for the different kinds of relationships, discussed earlier in Section 3.1.4, is shown in Figure 3.5. The symbols shown in Figure 3.5 are the arrowheads of the lines representing the different kind of relationships.

-->	Dependency
△	Generalization
—	Association
◇	Aggregation
◆	Composition

FIGURE 3.5 Diagramming notation for the different kinds of relationships

Multiplicity (defined earlier in Section 3.1.4.3) of an association is indicated on class diagrams by the UML notation as shown in Figure 3.6. The associations shown in Figure 3.6 are the generic associations and the words '*one*' and '*many*' would also have been sufficient to define their multiplicities (see Section 3.1.4.3). The formal notation as shown in Figure 3.6 should, however, be used in the class diagram, since it promotes an accurate definition of the multiplicity. Multiplicity must be read separately in the opposite direction for each association. In Figure 3.6 the association named *Asso I* is an example of a *one-to-one* association, i.e. each instance of *Class A* can be linked to exactly one other instance of *Class B*. *Asso II* is a *one-to-many* association, which implies that the association structure allows that an instance of *Class A* can be linked to 0 or more instances of *Class B*. When the association is read from the opposite direction, it implies that each instance of *Class B* must be linked to exactly one object of *Class A*. *Asso III* is a *many-to-many* association, which implies that an instance of *Class A* can be linked to 0 or more instances of *Class B* and, inversely, each instance of *Class B* can have 0 or more instances of *Class A* assigned to them. The generic associations can be refined by specifying relationship rules, i.e. placing constraints. The association shown in Figure 3.7 is a special kind of *many-to-many* association, where each instance of *Class A* can be linked to one or more instances of *Class B.* A constraint is placed here, viz. that there should be at least one instance of *Class B* assigned to the instance of *Class A* taking part in the association. The specified domain for a constraint can be a range of values such as 0..3, 1..5, 3..*; or discrete values such as 3,7,20; or combinations of the two, for example 1,5,8...*.

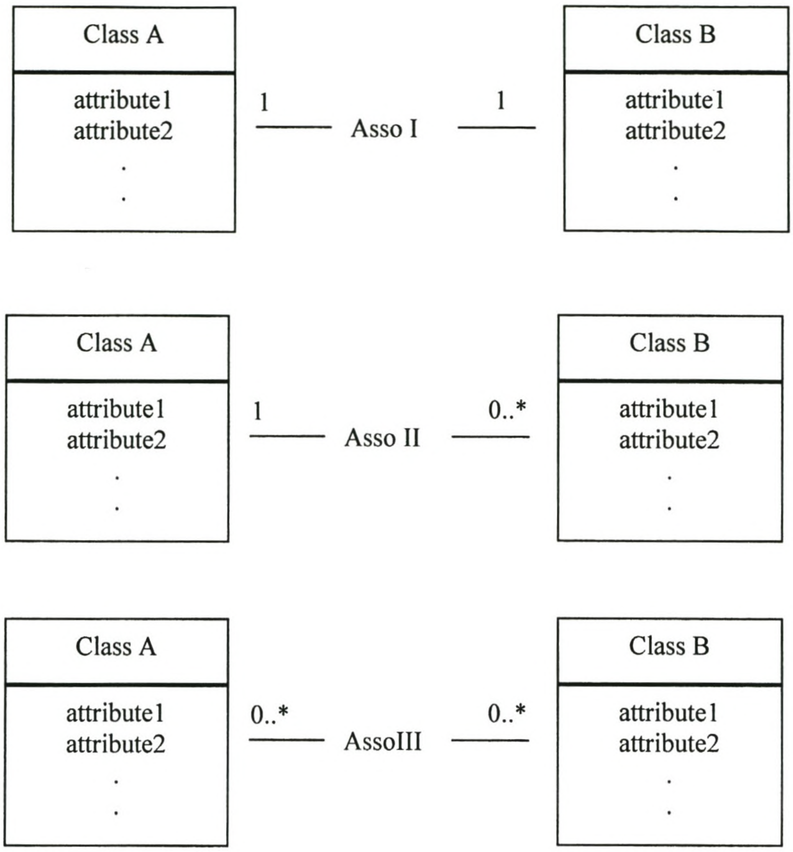


FIGURE 3.6 Multiplicities of generic associations

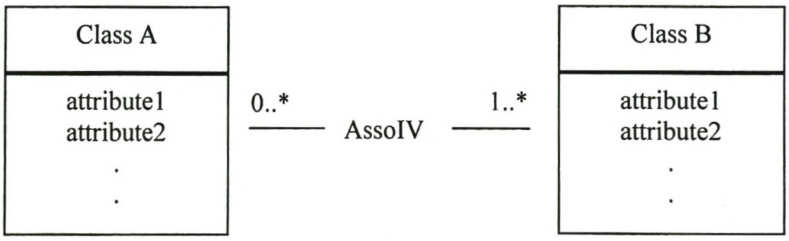


FIGURE 3.7 Multiplicity of a specialised *many-to-many* association

3.2.3 Object diagram

An object diagram, also referred to as an instance diagram, is a diagram that shows a set of objects and their relationships at a point in time - i.e. a snapshot of the objects in a system at that given moment in time. An object diagram is essentially an instance of a class diagram or the static part of an interaction diagram (see next section). Object diagrams can therefore be used to model the static design view of a system.

The object name and corresponding class name should be specified in the header compartment of an object diagram, separated with a colon and underlined. The object attributes are qualified in the object diagram, i.e. the attribute names with their corresponding attribute values are listed (see Figure 3.8).

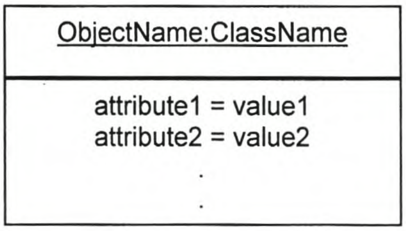


FIGURE 3.8 Object diagram

The third compartment containing the operations (see Section 3.2.2) can be left out on the object diagram, since an object diagram is used to capture the static state of the objects in the system.

3.2.4 Sequence diagram

An interaction sequence diagram (or simply referred to as a sequence diagram) shows the interaction between objects arranged in a time sequence. A variation of the sequence diagram is the collaboration diagram. The collaboration diagram basically differs from the sequence diagram in that it does not show the time sequence of the object interaction.

The most basic application of a sequence diagram is to represent the detailed object interaction that occurs for one use case. A sequence diagram of this kind can be seen as an expansion of a use case to the lowest possible level of detail. Interaction diagrams can be drawn during the analysis and also design phases of project development. In the design phase the diagrams will contain more details than in the analysis phase, and will show details such as message signatures and contain design objects such as components of the user interface.

The UML notation of the sequence diagram is shown in Figure 3.9. The vertical dimension represents time and all objects or classes involved in the interaction are spread horizontally across the diagram. Each object or class is represented by a vertical dashed line, called a *lifeline,* with an object symbol at the top. A message is shown by a solid horizontal arrow from one lifeline to another and is labelled with the message name. When a message is sent to an object, it invokes an operation of that object (see Section 3.1.5). Once a message is received, the operation that has been invoked begins to execute. The period of time during which an operation executes is known as an *activation* and is shown on the sequence diagram by a rectangular block placed along the lifeline. *Iteration*, i.e. a message being sent repeatedly, is indicated on the sequence diagram by a * preceding the message name. Conditional brackets can also be placed at messages where *branching* occurs. Object creation (instantiation) is shown on the diagram by a message arrow drawn with its arrowhead pointing directly to the object symbol at the top of the lifeline (this will invoke the object's constructor operation). Object destruction is indicated by a large X on the lifeline at the point in the interaction when the object is destroyed.

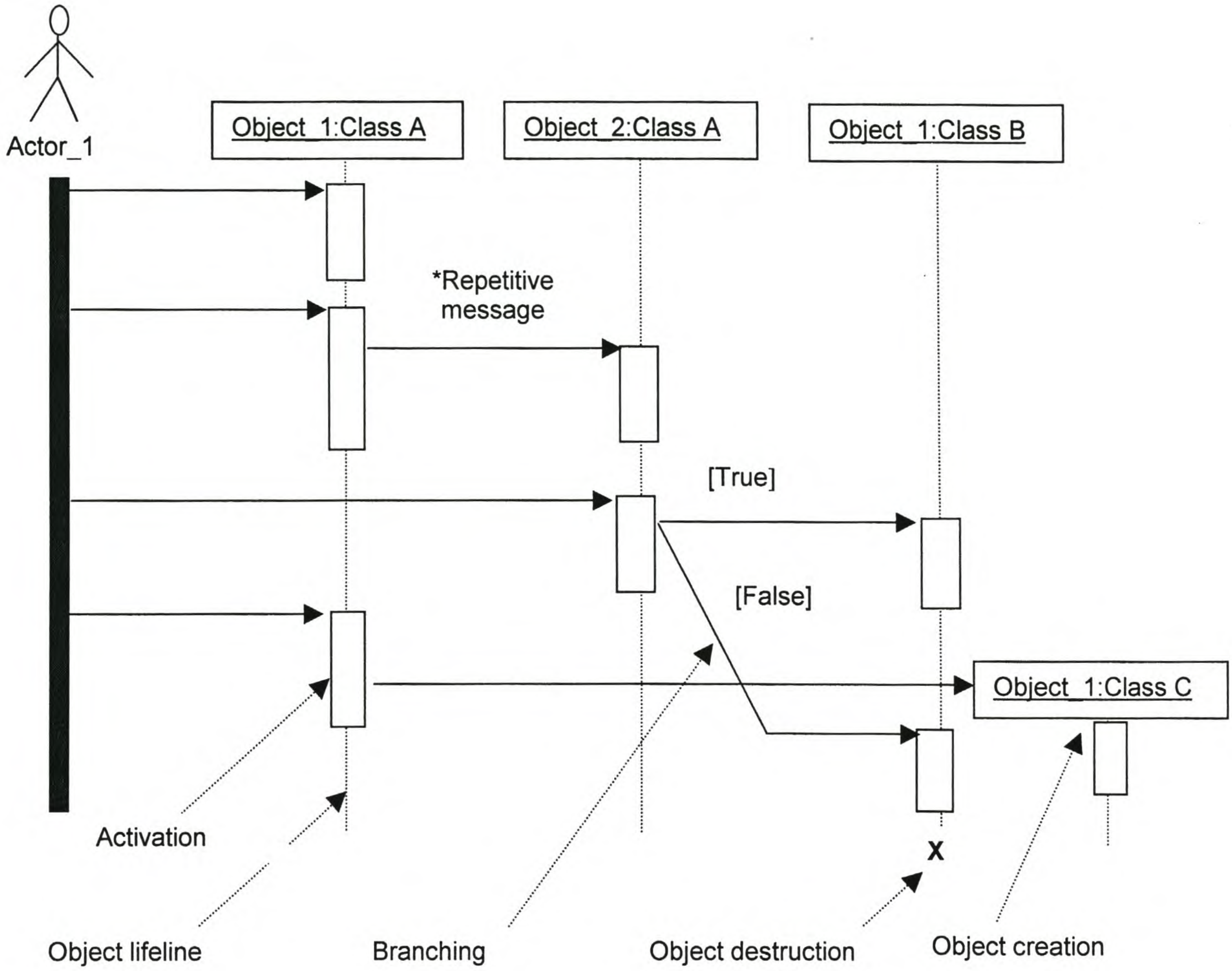


FIGURE 3.9 Sequence diagram

3.2.5 Activity diagram

An activity diagram is essentially a flowchart showing the flow of control from activity to activity within an object. In an activity diagram most states are action states (also called activities), each of which represents the execution of an operation or the evaluation of some expression. Activity diagrams are ideal to model the internal workings of an object's operations and the UML notation used (see Figure 3.10) greatly resembles the earlier flowcharts that were used to describe the flow in procedures.

The initial action state is indicated by a solid filled circle (see Figure 3.10). The action states are represented by rectangles, and decision points by diamond-shaped decision icons. Decision points must have one or more input arrows from preceding action state(s). There are usually two output arrows one for *true*, i.e. if the condition holds, and one for *false*. Decision points can be nested one after the other to represent complex decisions. The final state is shown by a bull's-eye symbol.

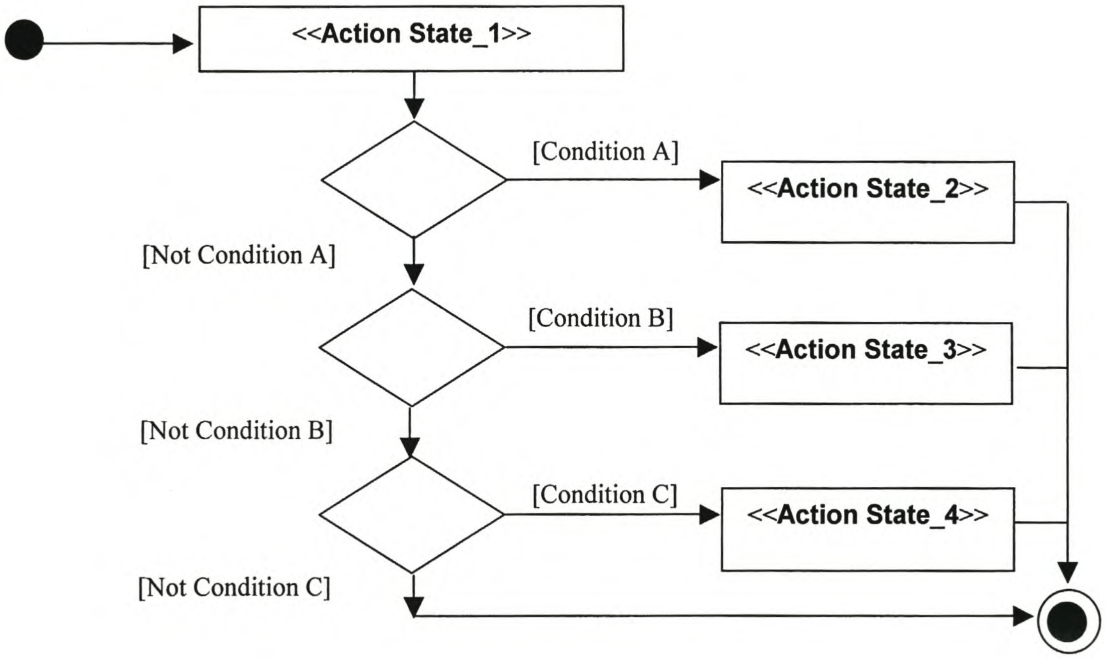


FIGURE 3.10 Activity diagram

3.2.6 Packages

A package is a general-purpose mechanism for organising elements into groups. The elements contained in the package can be classes, interfaces, collaborations, UML diagrams, other packages and even complete systems and subsystems. Graphically, a package is rendered as a tabbed folder with the package name as indicated in Figure 3.11.

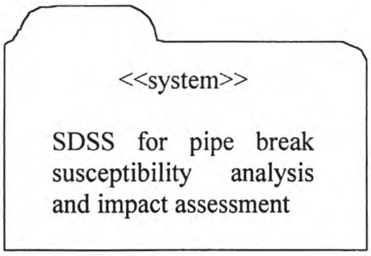

<<system>>
SDSS for pipe break susceptibility analysis and impact assessment

FIGURE 3.11 Package

There are mainly two kinds of relationships between packages: *import* dependency, used to import into one package elements exported from another, and *generalisation*, used to specify families of packages. The public parts of a package are called its exports. The exports of a package are visible to the contents of those packages that explicitly import the package. Generalisation among packages, where general elements are inherited by specialised packages, is very similar to generalisation among classes. Packages can also have the 'is-composed-of' relationship (see Section 3.1.4.3), where one package is composed of various other packages.

UML defines five standard stereotypes to indicate the type of package that is used. In this dissertation only the <<subsystem>> and <<system>> stereotypes will be used to indicate a package that represents an independent part of the entire system being modelled and a package that represents the entire system being modelled.

3.3 SUMMARY

In this chapter object-oriented modelling concepts have been discussed, such as: classes, object attributes and operations, object relationships, object encapsulation, object interaction, polymorphism and collection objects. The designs of the SDSS and its subsystems will be based on these concepts. Object relationships will play a significant role in the pipe break susceptibility analysis and impact assessment models. UML diagramming techniques have also been described in this chapter. UML diagrams will be used in this dissertation to document the user requirements and system designs. The standardised UML documentation method, should promote better communication in subsequent system development work. The next chapter presents the findings of a literature study of pipe break causes.

CHAPTER 4: PIPE BREAK CAUSES

This chapter is based on a literature study of known causes of pipe breaks. Since the information stems mainly from North American and European sources, some of the pipe break causes may not be relevant to South African water distribution systems. The pipe break susceptibility analysis model to be developed and incorporated in the SDSS will focus mainly on those pipe break causes that are relevant to the specific study area of this dissertation (see Chapter 7) and for which there are sufficient digital data available for the design and testing of the model. The pipe break susceptibility analysis model, which is thoroughly documented using UML, can be modified and extended to eventually support all the pipe break causes that will be discussed in this chapter.

Water mains are designed to withstand anticipated internal and external forces. However, structural failure can occur if the actual forces exceed the structural strength of the pipe material owing to poor design, deterioration of pipe strength or unanticipated forces (O'Day 1982; Ciottoni 1983). Poorly designed pipe sections in a water distribution system would normally be detected and corrected after the first couple of years in use. The pipe break causes that will be discussed in this chapter will therefore focus on only the last two categories, viz. pipe strength deterioration (also referred to as pipe wall deterioration) and unanticipated forces.

Pipe break seldom occurs as the result of a single cause; there is usually a combination of pipe breaking causes responsible and the final one determines the timing of the break (Morris 1967; Kottmann 1978; Ciottoni 1983; O'Day 1983). The pipe break causes will now be discussed in the sections that follow. Some of these causes and certain combinations of them will be modelled in Chapter 6 where the pipe break susceptibility analysis model will be described.

4.1 AGE OF PIPE

Extreme age does not necessarily indicate a poor condition of the pipe (Arnold 1960) and a facility does not deteriorate just because of age; rather it deteriorates due to corrosion and wear and tear from use (O'Day 1983). Arnold (1960), Ciottoni (1983) and O'Day (1983) all have studied the pipe breaks of the Philadelphia, Pa. water distribution system and found that there is no definite relationship between age and break rate. They reported on some more than 100-year-old mains that were still rendering satisfactory service with comparatively few breaks, whereas other 50-57-year-old mains have had frequent breaks. In a study in England of the Severn-Trent Water Authority's experience with main breaks, it has been reported that age is not a good indicator of condition, because certain pipes installed since 1952 have had break rates much higher than mains installed from 1890 to 1920. This phenomenon that newer pipes have break rates higher than older pipes was also found in an analysis of the water main breaks in Manhattan, New York City (O'Day 1982).

O'Day (1983) reports of a regression analysis that was conducted on age versus break rate of the pipes in Philadelphia and found that there was no relationship, since the coefficient of determination was very low viz. 0.008. This indicates that age is not useful for predicting the break rate for individual mains in a study area. Similar regression analyses were conducted by the US Army Corps of Engineers on the water mains of Binghamton, N.Y. The results showed a similar low coefficient of determination viz. 0.162, which indicates that age alone is a poor indicator of main break patterns (O'Day 1982).

O'Day (1983) warns that care must be taken to avoid over-generalisation when using age as a measure to predict pipe failure. One year in a severely corrosive environment can be equivalent to 30-50 years in a mildly corrosive environment. The soil conditions, the presence of stray direct currents and many other local factors are more important to consider than age when analysing break rates (Ciottoni 1983; Goulter & Kazemi 1988).

Age of pipe, however, should not totally be excluded in pipe break susceptibility analysis. O'Day (1983) has found from the Philadelphia study that a more significant coefficient of determination can be obtained when analysing the average break rate by age for 5-year age groups. The analysis showed that on average, older mains have greater break rates than younger mains. The keyword is average and that many other factors should be taken into account when predicting specific break rates for individual mains. Goulter and Kazemi (1989) have found from a detailed examination of the pipe failures in the City of Winnipeg that most of the failures occurred in pipes between 11 and 30 years old. Also, when the proportion of new pipes (i.e. renewed portions of the network) in an area increases, the overall rate of pipe failure in the region goes down, since new pipes in general fail less frequently than old pipes.

4.2 CORROSION

Corrosion has been defined by the United States National Association of Corrosion Engineers (NACE) as: 'the degradation of a material by reaction with its environment'.

Corrosion is often considered to be a phenomenon restricted to metals only, but this wider definition implies in fact that the various different degradation reactions on other materials such as concrete, plastic, etc. are all part of the corrosion process (Duligal 1990). The term *aggression*, however, is sometimes used to refer to the specific corrosion processes of concrete structure. In this chapter this distinction will also be made, viz. corrosion of metal pipes will be discussed further in this section and then in Section 4.3 aggression on concrete and asbestos-cement pipes will be discussed.

Corrosion of cast-iron pipes is a process that deteriorates the pipe wall by an electrochemical reaction (graphitisation) between the pipe metal and its environment; the pipe loses its ferritic constituent, leaving behind the graphite (O'Day 1982; Böhm 1993). The pipes usually then break because of the internal water pressure which cannot be contained by the corroded walls any longer or any other unusual strain that may occur (Arnold 1960; Böhm 1993). Various studies conducted by Kettler and Goulter (1985) showed that the cast-iron pipes have a decreasing trend in pipe failure rate with increase in diameter. The reason therefore is mainly that the large diameter pipes have a larger wall thickness than the small diameter pipes and can thus offer better resistance to corrosion. Corrosion of cast-iron pipes can occur internally or externally.

Internal corrosion is caused by corrosive water flowing inside the pipe, corroding the pipe walls from the inside. Normally, waters low in pH or high in dissolved oxygen, total solids and carbon dioxide will corrode most ferrous metals and can lead to failure (Morris 1967). If a failure, however, does not occur then, as the process continues, the products of corrosion (tubercules), build up, reducing the carrying capacity of the main. Although this presents operational problems, tuberculation is not a major cause of water main breaks (Morris 1967; Ciottoni 1983).

External corrosion is the deterioration of the pipe walls where the metal is eaten away from the outside, usually in concentrated locations, thus producing a thin wall or complete breakthrough in small areas (Arnold 1960; Ciottoni 1983). The two major sources of external corrosion are galvanic reaction in corrosive soils and stray-current electrolysis (Arnold 1960; Morris 1967; O'Day 1982; Ciottoni 1983). They will be discussed in the sections that follow.

4.2.1 Galvanic reaction in corrosive soils

Cast-iron pipes can corrode as the result of a galvanic reaction between the metal pipe walls and its surrounding corrosive soil. Galvanic corrosion is an electrochemical reaction, as found in a common dry-cell battery (see Duligal 1990, for a detailed descriptions of the chemical reactions). Basically the cell is made up of an anode (negative), a cathode (positive) and an electrolyte (water with some dissolved metallic salts). A current flows in the electrolyte from anode to cathode. If they are connected by a metal conductor, the current will flow back to the cathode to complete the current. In the field the pipe wall will act as the conductor and moisture in the ground will act as the electrolyte, if dissolved salts are present.

For a soil to be considered as corrosive, there should be a potential electrolyte present to transport the ions during the galvanic corrosion process. These soils are characterised by their high moisture content and high content of dissolved salts. Areas on a well-drained hill with very little moisture, for example, cannot act as potential electrolytes. Soils that are thoroughly leached out will also not be able to function as an electrolyte. Residual soils, especially those that are formed over limestone or shale, will normally be saturated with mineral salts, and can act as an electrolyte if the moisture content is also satisfactory (Morris 1967).

In general, soils that have a high clay constituent can be classified as corrosive, because of their tendency for moisture retention and electrical conduction (Hudak, Sadler & Hunter 1998). O'Day (1982) refers to corrosion rate studies conducted on ductile cast-iron pipe samples laid into six different soil types. The high corrosion potential of clay soils has been confirmed by these corrosion rate studies, since the tests showed that the pipe sample in the clay soil had a weight loss of 4.2 kg/m^2 contact area in 8 years, whereas the pipe sample in the sandy loam soil had a weight loss of only 1.5 kg/m^2 contact area even after 14 years. Acid soils with high organic substances are also found to be highly corrosive (O'Day 1982; Böhm 1993).

4.2.2 Stray-current electrolysis

Stray-current electrolysis is an electrolytic corrosion caused by stray direct electrical current nearby metallic pipes. The ground acts as the pathway for return flow and the 'stray current' follows the path of least resistance and may therefore follow a ferrous metal pipe, if there is one in the vicinity. At the point of departure from the pipe, metal ions will be extracted forming a pit of corrosion (Morris 1967; Böhm 1993). It is found that 1 amp of current can remove 9 kg of metal in a single year. Extensive damage was caused in certain water distribution systems where as much as 600 amp of current was measured travelling on the underground piping system (Arnold 1960; Ciottoni 1983).

Stray direct currents leaking from the old electric streetcar systems were mainly responsible for this type of corrosion (Arnold 1960; Morris 1967; O'Day 1982; Böhm 1993). The problem is still relevant in cities where there has been a partial abandonment of the streetcar system. Often the abandoned tracts, when they are no longer used for streetcars, are paved over and used as a return path for the current supplying operating tracks. These old rails, being paved over, are not properly maintained and open joints frequently develop, resulting in discharge of large quantities of current to the ground and underground pipes (Arnold 1960).

Certain bus systems today are still using direct current and can be a potential stray-current electrolysis source. Grounding systems (especially when directly grounded to the water distribution system) for welding workshops, telephones, radios, television and other electrical equipment, and rectifiers installed to protect utilities or structures from corrosion are also sources of direct current which can corrode metal pipes as the result of stray-current electrolysis (Morris 1967; Böhm 1993).

4.3 AGGRESSION

Aggression is a process that attacks the cement matrix of concrete structures, dissolving the free lime, calcium carbonate, as well as aluminates and silicates, causing eventual material collapse (AC Pipes 1994b; Mackintosh 1999). This occurs when the water or soil that is in contact with the concrete structures is either soft, i.e. low in calcium, or is acidic with a respectively low pH or has a high sulphate (SO_4) content (Whitlow 1993; AC Pipes 1994b; Allen 1998).

Asbestos-cement pipes and other cementitious pipes such as pre-stressed concrete pipes will deteriorate after long exposure in an aggressive environment. Asbestos-cement pipes consist of a tightly bound mixture of approximately 85% Portland cement and 15% asbestos fibres. The asbestos ingredient is a strong inert material that can offer great resistance against aggression. Asbestos-cement pipes are therefore more resistant to aggression than other cementitious pipes, but will eventually also deteriorate in severely aggressive environments (AC Pipes 1994b). Large diameter pipes have a larger wall thickness than the small diameter pipes and can thus in general offer better resistance to aggression than small diameter pipes (as was the case with the corrosion of cast-iron pipes, discussed in Section 4.2).

Aggression is a slow process that can take many years and often the final pipe breakage is caused as the result of the deteriorated pipe strength in combination with some other factor such as a sudden increase in pressure. The deterioration of cementitious pipe walls caused by aggression may be either internal or external. Internal aggression is caused by aggressive water that is flowing in the pipe itself, while external aggression is caused by aggressive ground water or aggressive soil that comes into contact with the pipe. The characteristics of aggressive water and soil, viz. soft, acidic and high sulphate content, will now be discussed.

4.3.1 Soft conditions

Soft waters and soils have a low calcium content. The hardness of water/soil is measured in mg/l as $CaCO_3$ and can be classified as shown in Table 4.1

TABLE 4.1 Hardness classification of water/soil
(adapted from Twort, Hoather & Law 1974)

mg/l as $CaCO_3$	Hardness of water/soil
0-50	soft
50-100	moderately soft
100-150	slightly hard
150-200	moderately hard
200-300	hard
>300	very hard

Cementitious products, including asbestos-cement pipes, are susceptible to attack from soft water or soil conditions. Under these conditions the lime hydrate ($Ca(OH)_2$) is leached out of the binding agent in the asbestos-cement, resulting in a decrease in effective pipe wall thickness (AC Pipes 1994b). The recommended method to be used to determine the potential influence of such conditions is the Langelier's Index which is defined as follows:

$$LI = pH - pH_s \tag{4.1}$$

Where:

LI = Langelier index of water or soil

pH = actual pH of water or soil

pH_s = calculated saturation pH based on the alkaline and carbonate hardness of the water or soil

Should the index be positive, no soft water (or soil) attack will take place. When the index is negative, the saturation pH (pH_s) value is higher than the actual pH value, and therefore the water (or soil) is under-saturated with $CaCO_3$ and it will tend to dissolve existing sources of calcium carbonate (Twort, Hoather & Law 1974; AC Pipes 1994b). For a calculated Langelier's Index of between 0 and –2, a single bitumen dip offers sufficient protection, and between –2 and –4, a double bitumen dip provides the necessary protection. Asbestos-cement pipes are generally not suitable for use should the Langelier's Index be less than –4 (AC Pipes 1994b).

Problems suffered from internal aggression in water pipes as the result of soft water have been reported by the Swiss project aid scheme, Helvetas, which has laid hundreds of kilometres of asbestos-cement pipe in Cameroon. Protective bitumen coatings were necessary to counteract the aggression (Stolz 1978).

Aggression damage in water pipes is also encountered in many South African water distribution systems, where approximately forty percent of the surface water and all the surface waters of Lesotho are soft (0-20mg/l as $CaCO_3$).

Virtually all of the groundwater on the southern and eastern fringes of South Africa (up to approximately 200 km inland) are also soft. These surface and groundwaters are also acidic with low pH values (4.0 – 7.0), which further aggravates the aggression damage (see Section 4.3.2) (Mackintosh 1999).

Furthermore, clay soils have the tendency to absorb calcium via cation exchange (Craig 1987; Whitlow 1993), when it comes into contact with substances with high concentrations of calcium, such as in concrete pipes and asbestos-cement pipes. This can result in lime to be leached out of the pipe walls.

4.3.2 Acidic conditions

Ordinary Portland cement is known to be alkaline resistant (pH>7), but is not recommended for use in acidic environments when pH≤6 (Whitlow 1993). This can be confirmed by a study conducted in the Netherlands of asbestos-cement mains that were badly deteriorated after about 30 years in a slightly acidic soil with a pH of 6 (Morris 1967). The surface water of the south-western Cape region of South Africa is known to be extremely aggressive, since it is soft and acidic with pH values that can get as low as 4 as the result of the presence of dissolved humic substances. Most municipalities in these areas stabilise the water by adding lime ($Ca(OH)_2$) to it, or they stabilise the water using the newer limestone stabilisation process before releasing it into the pipe network (Mackintosh 1999).

4.3.3 High sulphate content

Water and soils can also be classified as aggressive when the sulphate concentration is high (>150 parts per million) (Allen 1998). The presence of calcium sulphate (and to a lesser extent, sodium and magnesium sulphates) is fairly common in many clay soils. These sulphates often get dissolved from the clays in the form of sulphate ions (Twort, Hoather & Law 1974; Whitlow 1993). Sulphate ions have a serious deleterious effect on certain constituents of Portland cement, causing expansion and disruption of the cement paste Whitlow 1993; Allen 1998).

4.4 TEMPERATURE

Low temperature, high temperature and temperature differentials can induce unanticipated forces on a pipe which can cause pipe breakage, especially if the pipe material is already weakened by aggression/corrosion or the pipe bedding has been disturbed. Low temperature and temperature differentials can cause pipe breaks only in very cold regions - these two pipe break causes are therefore not relevant to South African water distribution systems.

4.4.1 Low temperature

In very cold regions such as in many parts of Europe and the USA, the low surface temperature will cause freezing of the water in soil, resulting in the formation of a frozen layer beneath the ground surface. The depth of the frozen layer is mainly dependent on surface temperature and duration of freezing conditions. In the United Kingdom it is unusual for the depth of penetration to exceed 0.5 m (Whitlow 1990). In middle Europe, however, during normal winters and up to 300 m above sea level, the frost depth can often reach 0.7 m (Böhm 1993).

Frost layers can cause frost heave, induce thawing effects or cause freezing of the water column inside the pipes. These three aspects of frost action and also how they can cause pipe breakage will now be discussed.

4.4.1.1 Frost heave

In cold conditions, when the surface temperature falls below 0°C, water in the soil will freeze causing an increase in volume and a vertical surface expansion which is termed frost heave. There are two components of frost heave. Firstly, as the porewater in the ground freezes, its volume increases by about 9 percent. In coarse-grained soils, such as gravels and coarse sands, this expansion can take place within the voids and so an increase in the overall volume is negligible. In clay soils, however, the voids are small and an overall soil expansion between 2 and 6 percent can occur, depending on the void ratio of the soil. Secondly, as water freezes, another phenomenon occurs. Since the vapour pressure of ice is much lower than that of water, a high degree of suction is created, which draws water upwards towards the frozen layer, where it freezes, thus increasing the ice crystals. The growth of the ice tends to be highly localised, resulting in the formation of ice lenses, which may increase the thickness of the frozen layer by up to 30 percent (Whitlow 1990; Böhm 1993).

The pressures set up by frost heave may be sufficient to lift road surfaces, paving and even lightly-loaded foundations (Whitlow 1990). Frost penetration forces, such as frost heaving, increase the external load on the pipe and can cause bending failure in situations where the bedding has been disturbed (O'Day 1982; Ciottoni 1983; Böhm 1993).

4.4.1.2 Thawing

When a frozen layer, especially one that contains ice lenses (see Section 4.4.1.1), undergoes rapid thawing, the soil near the surface will contain an excess of moisture. The additional moisture (mainly due to the melting of the ice lenses) will be unable to drain away through the still frozen soil below. The soil is therefore likely to be in an unstable state, especially in the case of a sloping ground surface. The shear strength and/or compressibility characteristics of the soil will be impaired. Additional surface loading (e.g. traffic load) can then cause damage and displacement to both the road base and pipe bedding (Whitlow 1990; Böhm 1993).

4.4.1.3 Freezing of water column

The freezing of a water column inside a pipe will induce severe circumferential tensile stresses as the result of the expansion caused by the phase transformation of water to ice. In most cases longitudinal ruptures will occur in the pipe wall (Böhm 1993). This freezing of the water column will occur very seldom, however, since the surface temperature must be extremely low for the coldness to penetrate the covering soil and pipe wall. Böhm (1993) refers to studies conducted on the isothermal lines around a 400 mm pipe laid at one metre depth. It has been found that the water temperature inside the pipe can still be as high as +5°C when the surface temperature is –14°C.

4.4.2 High temperature

High temperature causes air in the water to come out of solution, which can then accumulate in the form of air pockets. Air pockets can cause airbursts in pipes, which will be discussed in detail in Section 4.9.

4.4.3 Temperature differentials

The sudden chilling of the water and surrounding earth during winter months (especially at the beginning of the winter) can cause pipe shrinkage. This causes an increased longitudinal tensile (pulling) stress on the pipe, which can result in a break at a point already weakened by aggression/corrosion or some other factor (Arnold 1960; Morris 1967; Ciottoni 1983; Habibian 1992). An unusual, sudden rise in temperature in the winter can induce longitudinal compressive stresses in the pipe due to elongation of the pipe (Arnold 1960). In hot countries large temperature differentials seldom occur and a rise in temperate is less destructive than a drop, because a rise will induce compressive stresses, which most pipe material can withstand better than the tensile stresses resulting from a large drop in temperature.

The temperature-induced contractions/elongations, however, normally do not cause excessive stresses, because of the flexible couplings that are used between the pipes (O'Day 1982). These couplings usually have rubber sealers and spacer rings inside (see Figure 4.1) that provide some flexibility for relative movement between the pipes so that almost no axial stresses will be induced (Bazan 1991; AC Pipes 1995). Excessive tensile stresses can in fact only occur if structures come into contact with the pipe and restrict the contraction/elongations (O'Day 1982).

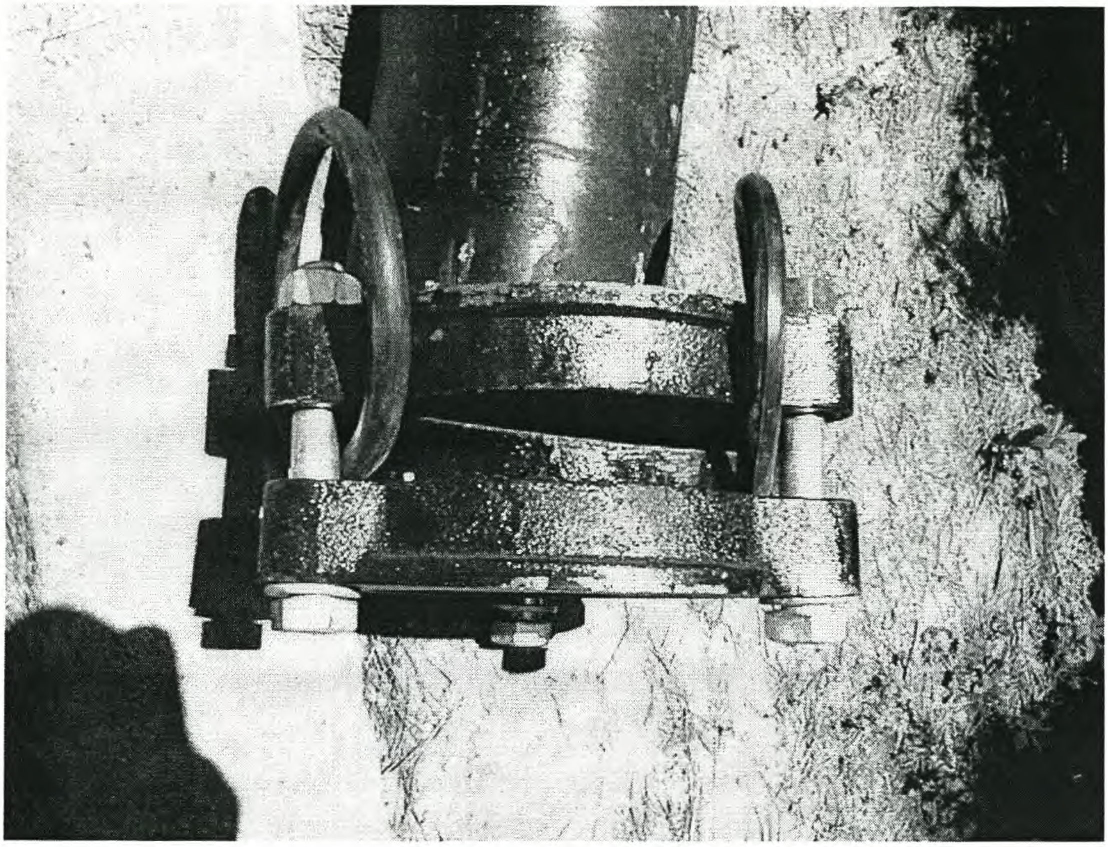


FIGURE 4.1 Flexible pipe coupling

4.5 TREES

Trees can provide various environmental benefits to a community, the most important of which are oxygen production (through the photosynthesis process), cleansing the air of dust particles, noise absorption and provision of shade (Morell 1992). Trees are valuable assets to commercial, private and public landscapes. They can be used to augment or modify many aesthetic, architectural, climatic, and engineering features in landscapes (Appleton *et al.* 1997) and trees can also increase property value (Morell 1992; Appleton *et al.* 1997).

Unfortunately, trees can also place a major burden on utility companies and municipalities, since they are often an obstruction to the urban infrastructure.

Trees can obstruct infrastructure (damage caused to water pipes will be discussed separately in detail) in many ways. The roots of some trees buckle and crack sidewalks and driveways (Gulick 1986; Skiera & Hennen 1986; Morell 1992). The obstruction of municipal and private sewer lines with tree roots is a widespread problem. Often these blockages originate when tree roots enter sewer pipes though cracked joints. At first thin roots will enter the crack and will then grow bigger inside the pipe (especially the roots of willow trees, which flourish alongside rivers, enter sewer pipes in this way). Grease and solids flowing in sewer pipes may then become entangled in these roots, ultimately producing backups (Groninger, Zedaker & Seiler 1997). Trees disrupt electrical and telephone communications when their branches come into contact with the utility lines. Lastly, trees can obstruct roadways and the vision of motorists (Morell 1992; Appleton *et al.* 1997). The latter two problems occur above the ground and tree pruning can be done to address them. The other problems mentioned are all tree-root related. Damage caused to water pipes by nearby trees is also tree-root related and will now be discussed.

Tree roots tend to follow the path of least resistance (Gulick 1986), i.e. in loose soil and avoiding obstacles; they also grow in the direction of where water (moisture) and nutrients are contained in the ground. Tree roots therefore often tend to grow in the sand bedding of a pipe, but since there are no nutrients in the sand, they will either leave the trench or continue to grow underneath/above and alongside the pipe in search of moisture and nutrients. Small cracks in the pipe and leaking valves may attract roots from further away to enter the trench in search of water. The roots, however, cannot enter the cracks, as was the case with the sewer pipes described earlier, since in a pressure pipe the water will discharge out of the crack at high velocity. Tree roots can, however, cause pipe break when the roots grow underneath the pipe, and then gradually increase in diameter over the years, thus lifting the pipe upwards at that point. A typical bending failure type of break will occur, viz. a straight, vertical and circumferential break (see Figure 4.2a and the close-up Figure 4.2b). In both figures the tree root causing the break can be seen (it stems from the tree shown in the background of Figure 4.2a).

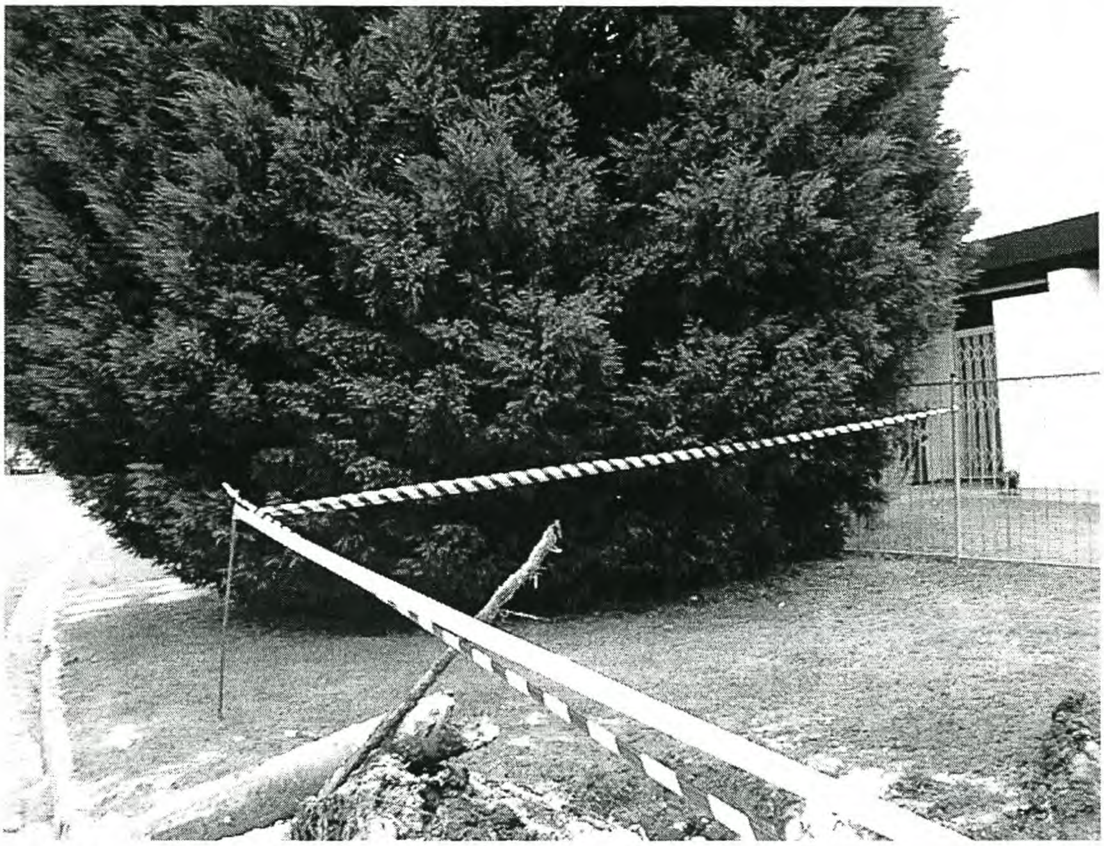


FIGURE 4.2a Pipe break caused by tree roots

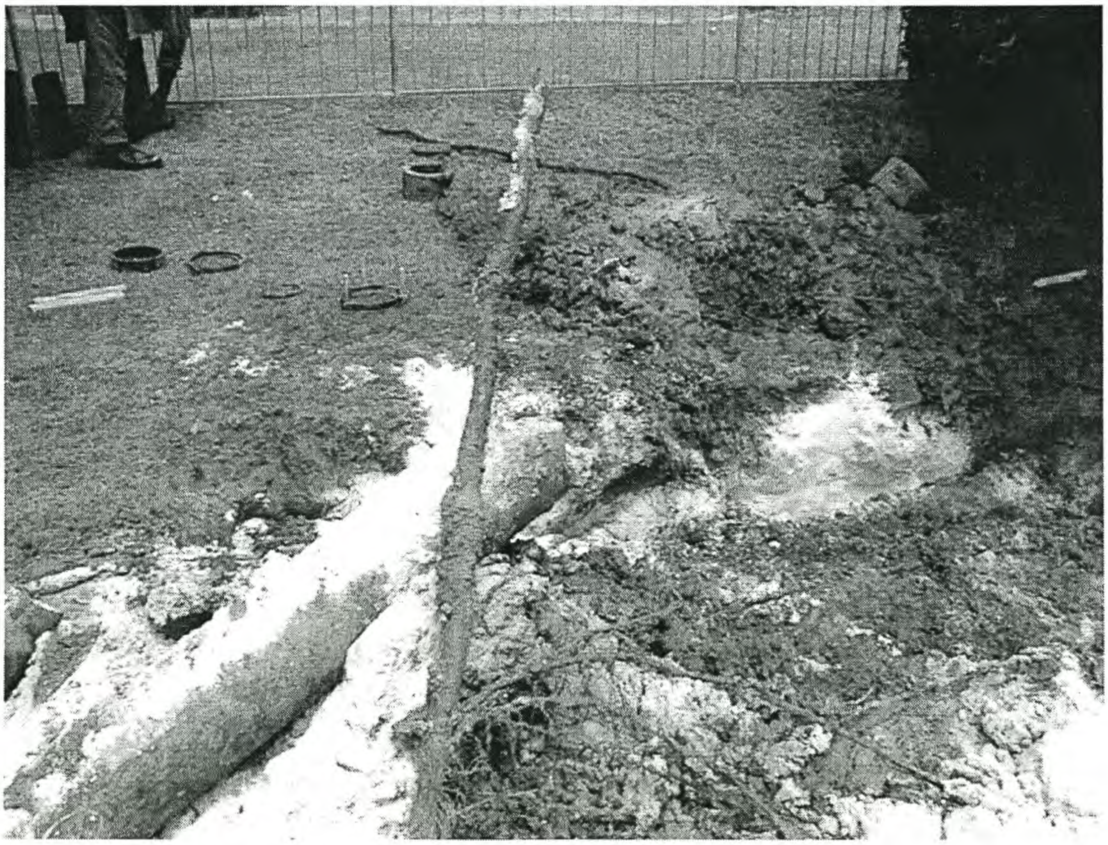


FIGURE 4.2b Pipe break caused by tree roots: Close-up

When analysing the pipe break susceptibility of a pipe as the result of nearby trees (discussed in more detail in Section 6.2.3), the following key points should be considered, viz. large trees have thicker roots than smaller trees and will have more strength to bend the pipe as described earlier. The further away the tree is from the pipe, the less destructive it can be to the pipe. Larger diameter pipes with thick walls (thus large section modulus) can resist the bending moment caused by the tree roots better than a small diameter pipe with a smaller wall thickness (and smaller section modulus). Cast-iron pipes, which have a much higher tensile strength than asbestos-cement and PVC pipes, will in most cases be able to withstand the forces induced by tree roots. Pipes laid underneath tar road surfaces will in general not be affected by tree roots. Unfortunately maintenance and emergency repair costs (also the inconvenience suffered by motorists) of pipes underneath the road surface are much higher than for those underneath the sidewalk. The laying of pipes beneath roads will therefore seldom be done, and usually only when the pipe must cross a road, or in confined business or industrial sectors where there is sometimes no sidewalk available.

The water distribution system, which is part of the infrastructure of a town, is an important asset for the community, but so are the trees, as mentioned above. In the confined urban environment, there is often competition for space (Morell 1992) and since both are equally important, engineers and foresters should work together to resolve the complicated conflict between infrastructure and trees. Maintenance personnel should take special care when conducting maintenance works on water pipes not to damage vital tree roots. Protecting the infrastructure from tree root damage is, however, more complicated, because some infrastructure is above the ground, while others are beneath the ground surface. Foresters have for example developed devices (such as the Deep Root Planter from The Deep Root Corporation in Westminster, California) to ‘train’ roots downward (away from the surface) and this way tree-root damage to surfaces such as pavements can successfully be reduced (Gulick 1986). This device, however, can be detrimental to the water pipes, since it encourages the roots to flourish at a deeper level, where the water pipes may be located. Careful root cutting (performed by a professional) can be done to protect water pipes from tree roots. Some trees, however, may decline after improper root cutting (Gulick 1986). When new trees are to be planted near a water pipe, the correct species must be decided upon. The rubber tree, for example, has very aggressive roots and will not be suitable near a water pipe, whereas an oak tree with its passive roots, will be more suitable. The amount of watering can also have an influence on the root system, since quick watering means that the water does not seep into the soil but encourages roots to flourish only in the upper portion of the soil - detrimental to the sidewalk pavement but beneficial for the lower-lying water pipes. The inverse, viz. deep watering, promotes deep root formation, which is detrimental to the water pipes but beneficial for the pavement (Gulick 1986).

A detailed pipe break susceptibility analysis, in view of the proximity of trees (see Section 6.2.3), and pipe break impact assessment (see Section 6.3) will help identify which pipe sections should receive higher priority for tree-root maintenance than others. The correct tree-root maintenance strategy must then be decided upon, i.e. a compromise must be made between the water pipe, the other infrastructure in that vicinity, and the well-being of the tree. Tree felling should only be considered at high-priority pipe sections (i.e. pipes with high break susceptibility as well as high break impact) and where no other solution could be found for the complicated and interrelated tree-infrastructure problem.

4.6 DIFFERENTIAL SETTLEMENT

If the ground shifts or settles unevenly, differential settlement occurs which can cause the water pipe to act as a beam subjected to stresses, which may exceed the tensile strength of the pipe material (Arnold 1960; Ciottoni 1983; Böhm 1993). A typical bending failure type of break will then occur, viz. a straight, vertical and circumferential break, which was also the type of break caused by tree roots (see Figure 4.2a and the close-up Figure 4.2b).

Differential settlement can be caused by volume changes in the supporting soil underneath the pipe. Expansive soils contain clay minerals that undergo significant volume changes with changes in moisture (Ciottoni 1983; Böhm 1993; Hudak, Sadler & Hunter 1998). Generally, expansive clays have a high plasticity index, reflecting a tendency to take up much water while in the plastic state. The shrink-swell potential of a soil is the percentage volume change it can undergo and can be classified as shown in Table 4.2.

TABLE 4.2 Shrink-swell potential classification of soil (adapted from Hudak, Sadler & Hunter 1998)

% Volume change	Shrink-swell potential
<3	low
3-6	moderate
6-9	high
>9	very high

Differential settlement can be caused by the gradual wearing away of the soil underneath the pipe as the result of a small leak (Ciottoni 1983). If the supporting soil underneath the pipe is of clay material, then a leak can further aggravate the problem, since the increase in moisture may cause further swelling in the clay (Hudak, Sadler & Hunter 1998). The wearing away of the soil around the pipe can also be caused by a sudden rainstorm or continuous rainfall, especially after a long dry period.

The level of the groundwater table and its seasonal variation can cause differential settlement, especially if the pipe is laid partly in clay and partly in sandy soils (Habibian 1992; Böhm 1993).

In certain parts of the world earthquakes can be the cause of differential settlements. The earth tremors and differential settlements occurring in mining subsidence are also relevant pipe break causes in South Africa (Brink 1979).

4.7 EXTERNAL IMPACT

The external impact on the pipes in a water distribution system as the result of the ever-increasing traffic load (i.e. an increase in traffic volume as well as axel load) can contribute to pipe breaks (Arnold 1960; Ciottoni 1983; Böhm 1993). It has been noted that some pipe breaks occurred as the result of potholes in the street that heavy vehicles drive into, setting up noticeable vibrations (Arnold 1960). The vibration caused by heavy traffic on an uneven road is estimated to reach a frequency of up to 40 Hz. If the vibration frequency matches the internal frequency of the soil that surrounds the pipe, then the pipe will vibrate together with the soil, causing a potential pipe break (Böhm 1993).

External impact on pipes can also be caused by the equipment and activities involved in new construction or the maintenance of other utilities in the vicinity of the pipe (Ciottoni 1983).

4.8 PRESSURE SURGES

Pressure surges (waves) can be induced in the pipe network by a sudden change in flow velocity as the result of, for example, a sudden closure of a valve, or a pump that is switched on or off. In a water pipe network, the Joukowsky surge seldom develops fully and will thus not be as destructive as in a long pipeline (Bazan 1991). The pressure waves, however, when amplified by air pockets (see Section 4.9.3.3) can then also be very destructive and cause pipe bursts (Böhm 1993; Kottmann 1995).

4.9 AIR POCKETS

Pipes made out of brittle material, grey cast iron, asbestos-cement and PVC repeatedly break for no obvious reason, causing municipalities major maintenance overheads and concern, since if the cause is unknown then further pipe breaks cannot really be prevented. The breaks only occur in pipe sections with low velocity and the pipes are often found ruptured into pieces, which could only have been caused by unusually high pressure.

The various studies conducted by Gandenberger in 1957 and then continued by Kottmann in the period 1980 to 1995 have finally thrown light on the matter by identifying entrapped air pockets as the major cause of these bursts. Air pockets as the pipe break cause, however, are often overlooked by the maintenance crews to this day, because they are not as obvious as most other pipe break causes (such as trees and traffic load) which can be identified more easily.

The next sections discuss air entertainment, air-pocket formation in a water distribution system and describe how air pockets can cause pipe break.

4.9.1 Air entrainment

Preventing air from entering the pipe will solve the whole airburst problem, but unfortunately the complete elimination of air is not possible. However, an understanding of how it may enter a pipe will assist in minimising its entry (Lescovich 1972).

Air consists mainly out of approximately 20% oxygen and 80% nitrogen. Both these gases are soluble in water, which means that the water flowing in the pipes will already have some air in solution when it starts flowing into reservoirs and pipe network. The solubility of air in water is a function of both pressure and temperature and a few other variables such as turbulence (Kottmann 1984; Van Vuuren 1990; Böhm 1993; AC Pipes 1994a). The solubility will increase when the pressure increases, but will decrease with rising temperature (Kottmann 1984). It is generally accepted that under atmospheric pressure, and at temperatures of 20°C, about 3% per volume of water consists of dissolved air (Van Vuuren 1990). Air will come out of solution when the pressure drops and/or when the temperature rises (Lescovich 1972; Kottmann 1984; Bazan 1991; Böhm 1993; Kottmann 1995). This is why in the late summer, when the temperature is at its maximum, the air-related bursts occur most often. Airbursts usually occur in the daytime when the system is in full operation and pressures are at a minimum (Kottmann & Schmitt 1980; Kottmann 1984; Kottmann 1995). Pressure drops and corresponding degassing can occur where a pipe diameter changes to a smaller diameter (Bazan 1991). The turbulence in the water at these pipe reduction points and also at other pipe junctions can also release air (Böhm 1993). It has also been found that on long ascending stretches the pressure will decrease gradually, releasing the air out of the solution (Lescovich 1972; Twort, Hoather & Law 1974).

Besides the dissolved air in water, air can also enter the system in the following ways (Lescovich 1972): Air is present in a pipe prior to filling and can often not be purged completely. This may happen right in the beginning during construction when the pipes are filled for the first time, during normal pipe cleansing maintenance, or pipe cleansing after a pipe break. The large amount of water that discharges out of a pipe break can cause negative pressure, which will suck air into the system. Air may be drawn into a pipe through leaky joints at zones of negative pressure, such as at points above the hydraulic gradient. Malfunctioning air-release valves and vacuum-breaking valves, as well as wrongly placed air-release valves at negative pressure zones situated above the hydraulic gradient, can suck air into the system. Lastly, air may be drawn in at intakes by the vortex action of pump suction.

4.9.2 Air-pocket formation

Once in the system and out of solution, the liberated air tends to collect at summits, where the air pockets will increase in size. Maximum air accumulation will occur at summits that are also high points relative to the hydraulic gradient (Lescovich 1972; Twort, Hoather & Law 1974; Bazan 1991; Böhm 1993; Department of Housing 1995), since these high points will then accordingly have low pressure that will promote degassing (see Section 4.9.1). Air pockets are often also located at geodetic low points (Kottmann & Schmitt 1980; Kottmann 1984; Department of Housing 1995; Kottmann 1995), provided the pressure there is not too high. Since air bubbles rise upwards as a result of the buoyant forces, they will get stuck to the pipe ceiling at these high and low points in the pipe network, causing pockets of air to accumulate there. However, air pockets will not occur at low points in the system, where the pressure is very high, since air release will then be suppressed (see Section 4.9.1) and the air will be forced back into solution again.

Air does not necessarily move forward with the water, but may move backwards up against the flow in a pipe that has a downward slope (Lescovich 1972; Twort, Hoather & Law 1974; Wisner, Mohsen & Kouwen 1975). If the flow velocity is high in the pipes, the air may be swept along with the flow, automatically purging the line. In most instances the forward movement of the water will force the bubbles into the downstream slope of the pipe. At some specific point, the buoyant and drag (velocity) forces become equal and the bubbles remain stationary (Kalinske & Bliss 1943; Lescovich 1972; Wisner, Mohsen & Kouwen 1975; Krug 1988; Bazan 1991). An accumulation of these stationary air bubbles will then form an air pocket in that section. The velocity required to move air down a pipe slope increases with pipe diameter and slope (Kalinske & Bliss 1943; Lescovich 1972; Wisner, Mohsen & Kouwen 1975). This sweeping velocity can be calculated with either the formula of Kalinske and Bliss or Wisner (as given in AC Pipes 1994a), but for a pipeline consisting of various high and low points, it is recommended rather to use the figures from the experiments conducted by Kottmann and Schmitt (1980). For the experiment they constructed a small-scale pressure tube consisting of three high points and two low points and found that the velocity must be between 0.2 and 0.3 m/s for air pockets to remain in the system. When the velocity exceeds 0.3, the air will be swept out of all high and low points.

Apart from the high and low points already mentioned, air pockets can also occur in the system at the following locations and under the following conditions (Lescovich 1972; Twort, Hoather & Law 1974; Vent-O-Mat 1995): if the flow velocity in the pipes of a section is less than the sweeping velocity, then air will not be swept along with the flow and air pockets can occur somewhere in these sections. In relatively shallow slopes, where the flow velocity equals the sweeping velocity, the buoyant and drag (velocity) forces become equal and the bubbles remain stationary. At these locations air pockets can occur. If the slope is relatively flat and long, an air pocket may extend most of the length of the run. Air pockets can also occur along the whole length of a rather flat but slightly ascending slope, where additional air will come out of solution as the pressure gradually drops (see Section 4.9.1), which will further enlarge the air pockets. A lessening of upgrade in the direction of flow can cause an accumulation of air.

Pipes with steep slopes are usually free from air pockets, since the air bubbles can easily rise with the flow or backwards up against the flow, so that air accumulation will occur somewhere else in the system (Kottmann 1995).

Lastly, there are two other points to consider that can contribute to air-pocket formation. Firstly, at pipe diameter reduction points, air will come out of solution because of the pressure drop and turbulence (see Section 4.9.1), and the liberated air may then be trapped before or after the reduction point. Normal pipe junctions where smaller pipes join in will undergo similar turbulence that can release air bubbles, which can accumulate to form air pockets. Secondly, turbulence and corresponding air-pocket formation can occur at blank ends, where a pipe is terminated by a blank flange or a valve (Vent-O-Mat 1995). When the pipe slopes upwards with the blank end at the top, the air and corresponding air pocket will rise and accumulate at the blank end, which should normally be strong enough to withstand the pressure surge that can occur (see Section 4.9.3). When the pipe slopes downwards with the blank end at the bottom, however, the air will rise from the blank end backwards and up the pipe. If the pipe slope is relatively flat, then equilibrium can be reached, as described earlier in this section, and an air pocket will form. This air pocket will then be located in the pipe somewhere in between the start node and blank end node, where it can cause a destructive pressure surge (see Section 4.9.3).

4.9.3 Pipe break caused by air pockets

Air pockets can induce three different types of pipe break mechanisms, viz. sudden air release under static conditions, water column collision under operating conditions and surge pressure amplification. The latter is considered to be the break mechanism that occurs most often of the three. The three break mechanisms will now be discussed.

4.9.3.1 Sudden air release under static conditions

Pipe couplings with their rubber sealers (see Figure 4.1) are usually watertight but not necessarily air tight. Should an air pocket be present in a pipe that is under static pressure, and at a certain instance one or more couplings release air under the applied pressure, the compressed air escapes almost instantaneously. The surrounding water that rushes into the created void, unable to escape through the watertight coupling, will produce a waterhammer effect (AC Pipes 1994a).

It has been found experimentally that pressure surges will also occur if the air exits through an opening which is not watertight, because of the difference in the flow characteristics of water and air. Once all air has been released, water will start flowing from both sides towards the opening. Because of the much higher density of the water, the flow rate through the opening will drop radically. This deceleration causes a pressure surge, which has the same effect as a sudden closure of a valve (Van Vuuren 1990).

Whether the air escapes through the watertight or the non-watertight opening, both cases can induce destructive pressure surges causing pieces of the pipe wall to be ripped off and shattered around the pipe bedding.

4.9.3.2 Water column collision under operating conditions

When an air pocket of considerable size occupies a certain part of a pipe in which water is flowing, surge pressures may be induced by the air pocket itself without the air actually escaping from the system. The mechanism can best be explained by means of the illustration (Figure 4.3). From continuity it follows that the flow in the pipe is equal through cross-sections A_0 and A_1. The flow velocity v_1 is thus greater than v_0. When the equation of Bernoulli is applied, it follows that pressure P_1 must be smaller than pressure P_0. As mentioned earlier, the amount of dissolved air in water is a function of both pressure and temperature. When the temperature is constant and the pressure decreases, as in the vicinity of cross-section A_1, more air will be liberated from the water and the size of the air pocket will increase. This will result in a further increase in velocity v_1 and a decrease in pressure P_1. The air pocket may eventually get so big that it will occupy the whole cross-section of the pipe for a short period of time, resulting in a momentary interruption of the flow. The collision of the two water columns will then cause a surge wave of non-negligible magnitude, which can be similarly destructive as the one caused by sudden air release (AC Pipes 1994a).

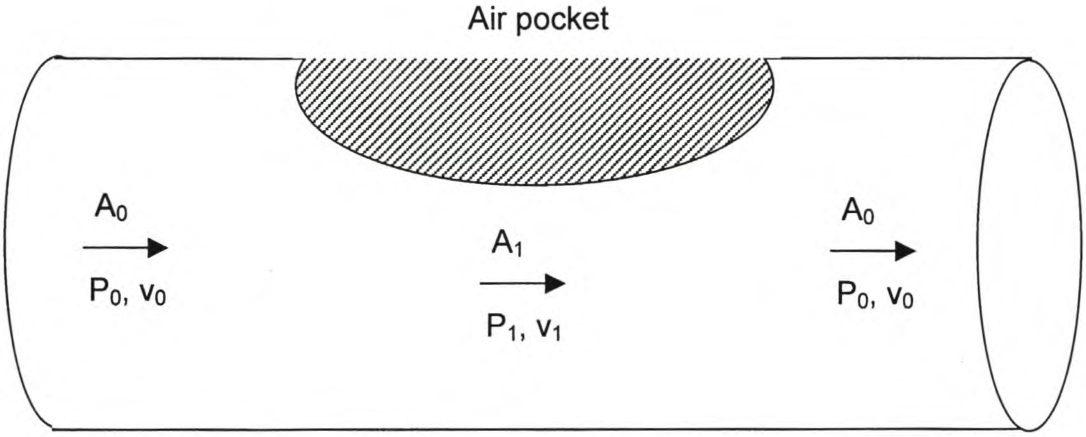


FIGURE 4.3 Pressure variations caused by entrapped air in a pressure pipe (adapted from AC Pipes 1994a)

At some low points in the system where air pockets may occur (see Section 4.9.2), the accumulation of silt, which is common at low points (Böhm 1993; Department of Housing 1995), can further reduce the cross-section of the pipe and thereby increase the chance of water column collision.

4.9.3.3 Surge pressure amplification

Surge pressure amplification is the most common pipe break mechanism resulting from entrapped air pockets. The conditions that give rise to surge pressure amplification build on the previous conditions as described in Section 4.9.3.2. As the water leaves the high-speed section underneath the air pocket (see Figure 4.3), it decelerates again to the speed it had in the full-flow section before the air pocket. This is because the cross-section before and after the air pocket is the same and the flow in the pipe follows the concepts of continuity. The deceleration is, however, only possible with a transfer from kinetic energy to heat by additional turbulence. This produces a turning water roller after the air pocket. Since the area of flow is extremely concentrated under the turning water roller, a high-speed section will be located there, where changes in the flow velocity initiated in the system (see Section 4.8) will be magnified. The water roller behind an air pocket thus acts as a pressure amplifier. Small pressure waves initiated in the system will be picked up and amplified by the roller, resulting in a sudden destructive pressure spike in the vicinity of the water roller. Since the duration of the spike is very short, approximately between 1/1000 and 1/100 of a second, it cannot be recorded on standard measurement devices. Only the result is visible, i.e. pipe bursts at high points, low points and sections with low incline. Pressure waves running with the flow will not be amplified. They will be reflected directly at the air pocket. Only pressure waves running against the flow get amplified at the roller and can cause a pipe burst. The type of rupture is almost always a longitudinal crack (see Figure 4.4), or a shell-type rupture (see Figure 4.5a,b). Very high pressure spikes can also rip the pieces apart, and they can be found shattered alongside the pipe (Kottmann & Schmitt 1980; Kottmann 1984; Böhm 1993; Kottmann 1995).

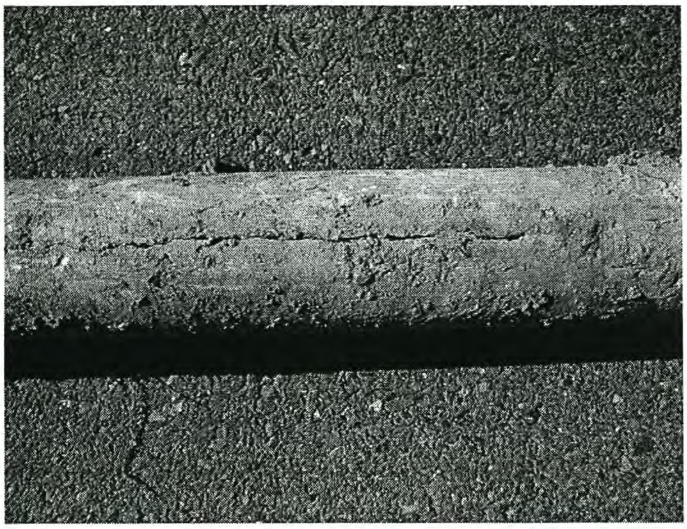


FIGURE 4.4 Longitudinal crack

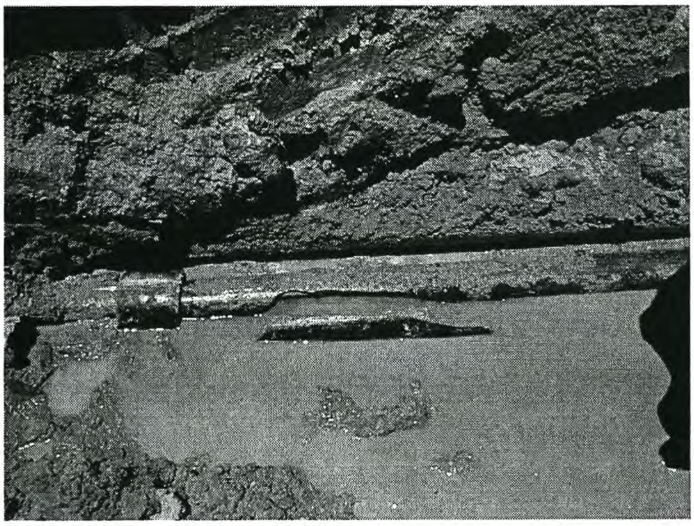


FIGURE 4.5a Shell-type rupture

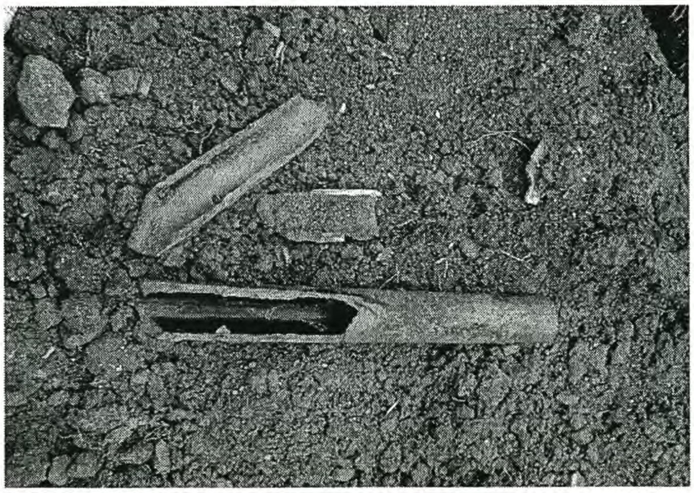


FIGURE 4.5b Shell-type rupture: Close-up

4.10 SUMMARY

This chapter has provided fundamental background on the causes of pipe breaks. Some of the pipe break causes outlined in this chapter are not relevant to South African water distribution systems. Low temperature effects (such as frost heave, thawing and the freezing of the water column), and temperature differentials can cause pipe breaks only in very cold regions - these two pipe break causes are therefore not relevant to South African water distribution systems. The pipe break causes relevant to the specific study area of this dissertation (see Chapter 7) and for which there are sufficient digital data available for the design and testing of the pipe break susceptibility models, will be incorporated in the knowledge base of the SDSS. The next chapter will outline the general user requirements and general design of the SDSS.

CHAPTER 5: GENERAL REQUIREMENTS ANALYSIS AND DESIGN OF A SPATIAL DECISION SUPPORT SYSTEM FOR PIPE BREAK SUSCEPTIBILITY ANALYSIS AND IMPACT ASSESSMENT

The general requirements analysis and design of the SDSS for pipe break susceptibility analysis and impact assessment are presented in this chapter. The requirements model and design will be based on the information systems theory and object-oriented modelling concepts discussed in Chapters 2 and 3. The basic user requirements of the SDSS regarding pipe break susceptibility analysis and impact assessment will first be outlined and then the requirements analysis and design of the general query, manipulation and display functionality of the SDSS will follow, which are needed to fulfil the basic user requirements of the system. The general data manipulation operations and the user interface will also be described and designed in this chapter.

5.1 BASIC USER REQUIREMENTS OF THE SPATIAL DECISION SUPPORT SYSTEM

As seen from a use case architectural modelling view (see Section 3.2), the SDSS for pipe break susceptibility analysis and impact assessment is composed of the subsystems as indicated by the UML packages shown in Figure 5.1. The use case architectural modelling view as shown in Figure 5.1 outlines the basic functionality of the SDSS as seen from the user's perspective. The SDSS user should thus be able to conduct a pipe break susceptibility analysis on the pipes in the water distribution system. The user should be able to carry out a pipe break impact assessment independently of the pipe break susceptibility analysis and, finally, conduct a combined pipe break susceptibility analysis and impact assessment where both the pipe break susceptibility and the impact are assessed collectively.

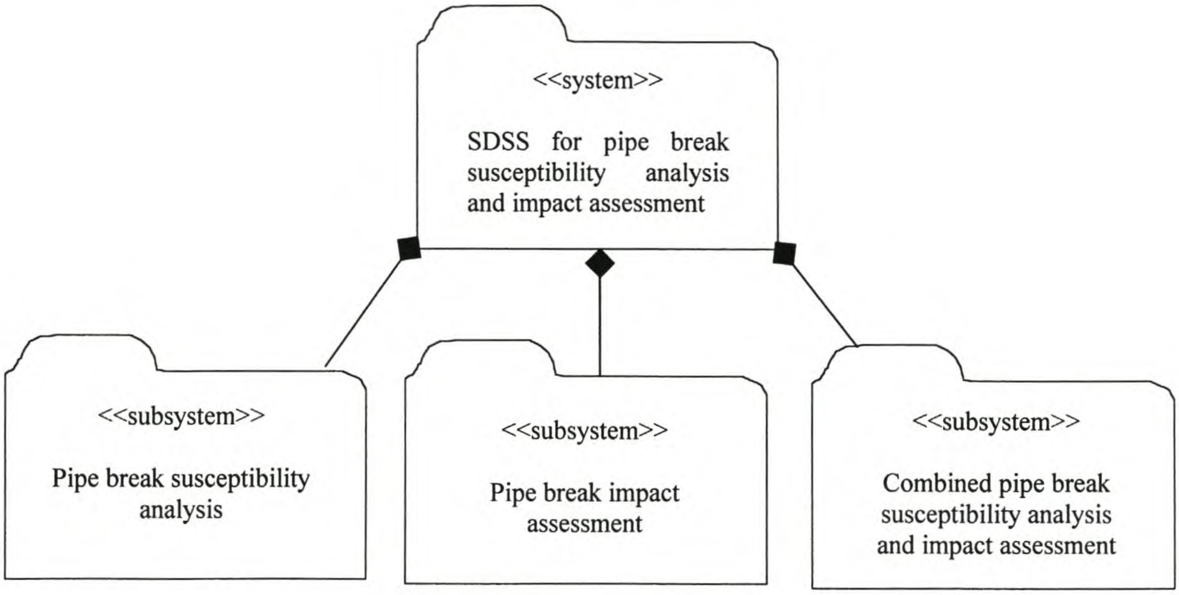

<<system>>
SDSS for pipe break susceptibility analysis and impact assessment
<<subsystem>>
Pipe break susceptibility analysis
<<subsystem>>
Pipe break impact assessment
<<subsystem>>
Combined pipe break susceptibility analysis and impact assessment

FIGURE 5.1 Use case architectural modelling view of the SDSS

The subsystems shown by the UML packages in Figure 5.1 can be further subdivided into subsystems for pipe break susceptibility due to specific pipe break causes and for the assessment of specific pipe break impacts. This subdivision will be made in Chapter 6.

5.2 GENERAL REQUIREMENTS ANALYSIS AND DESIGN OF THE SPATIAL DECISION SUPPORT SYSTEM

This section contains the requirements analysis and design of the general query, manipulation and display functionality of the SDSS. This general functionality, together with the specialised functionality for pipe break susceptibility analysis and impact assessment (see Chapter 6), should fulfil the basic user requirements of the SDSS (see Section 5.1).

5.2.1 Requirements analysis

The users of the SDSS, also referred to as actors of the system (see Section 3.2.1), can be grouped into the following four user types: a Systems Administrator for assigning user access rights, a SDSS Operator for data capturing work, a SDSS Analyst for data analysis, and a Public Works Administrator who will be responsible for the final decision-making.

The Systems Administrator should only have access to the administrative functions of the SDSS (see Figure 5.2) for managing the user names, passwords and corresponding user access rights.

The SDSS Operator, SDSS Analyst and the Publics Works Administrator should all have access to the basic navigational functions (see Figure 5.2) to be able to browse through the SDSS results.

The SDSS Operator should be able to specify and execute spatial queries (and spatial selections) and should have access to special spatial operations for interpolating elevations at points and access to hydraulic network flow operations to calculate the outflow from simulated pipe breaks. The SDSS Operator should further have access to basic object attribute data editing (manipulation) operations and should have access to thematic mapping functionality to assist him with the data-capturing work (thematic maps should give him a better overview of the spatial distribution of the data). The SDSS Operator will not have access to complicate data analysis functionality such as fuzzy inference and will also not have access to any graphing functionality, since these tasks are not within the scope of his job specification.

The SDSS Analyst should have access to the same spatial operations as the SDSS Operator and will additionally have access to object query functionality for specifying complex queries on the object attributes. The resulting objects from the queries will be stored into special collections and presented in a tree-like structure to have a good overview of all the query results. Long complex queries can be broken down into smaller and logically structured sub-queries, since the results from a previous query collection can be used as part of the antecedent of a new query. The SDSS Analyst should also be able to perform complex attribute manipulations, i.e. operations carried out on the objects whereby the attribute values can be edited and updated accordingly.

The SDSS Analyst should have access to a fuzzy controller (see Section 2.2.3) that is linked with the system, in order to perform fuzzy inference (see Section 2.2.2). Fuzzy inference will be necessary for the modelling of some complex aspects in both pipe break susceptibility analysis and pipe break impact assessment.

The SDSS Analyst should have access to the Multifactor Evaluation Process (MFEP), to combine results from various sub-queries. The MFEP will be discussed in detail in Section 5.3.7.

The SDSS Analyst should have access to a graphing subsystem where graphs of the SDSS results, as well as of the sorted result values can be drawn. The latter, which is a cumulative distribution graph, can then be examined visually to find an appropriate classification schema for the result values.

The SDSS Analyst should finally have access to thematic mapping functionality in order to create thematic maps based on a specified attribute. As can be seen from Figure 5.2, the SDSS Analyst has access to all the system functionality, except the administrative functions.

The Public Works Administrator should in addition to the basic navigational functionality, have access to the graph and thematic map subsystems to view the final summarised results.

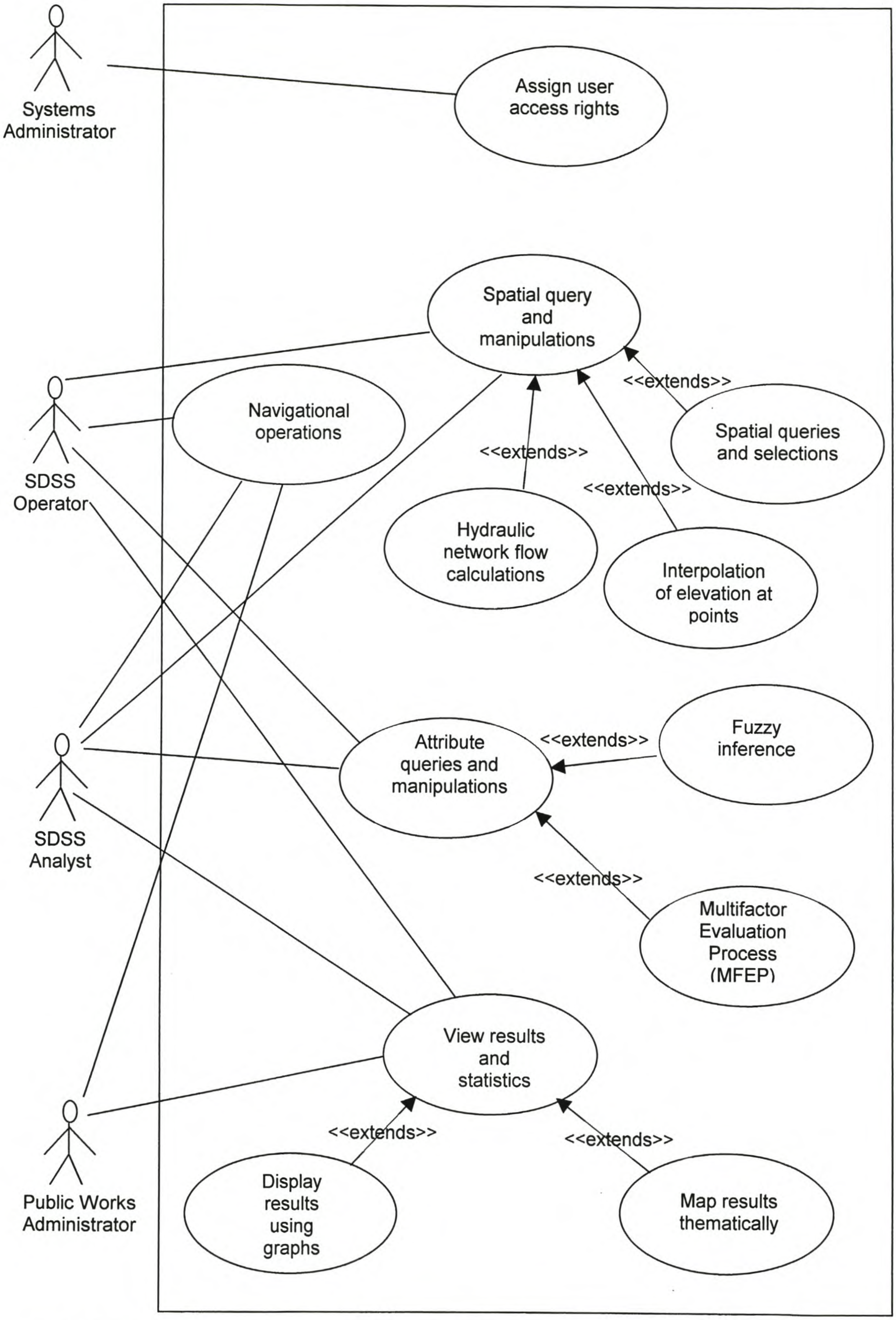

Systems Administrator
Assign user access rights
Spatial query and manipulations
SDSS Operator
Navigational operations
<<extends>>
Spatial queries and selections
<<extends>>
Hydraulic network flow calculations
<<extends>>
Interpolation of elevation at points
Fuzzy inference
<<extends>>
Attribute queries and manipulations
SDSS Analyst
<<extends>>
Multifactor Evaluation Process (MFEP)
View results and statistics
Public Works Administrator
<<extends>>
<<extends>>
Display results using graphs
Map results thematically

FIGURE 5.2 Use case diagram: General query, manipulation and display functionality of the SDSS

5.2.2 Design

As seen from a design architectural modelling view (see Section 3.2), the SDSS for pipe break susceptibility analysis and impact assessment can be grouped into the subsystems as indicated by the UML packages shown in Figure 5.3. The arrangement of the subsystems as shown in Figure 5.3 can provide a platform for the general query, manipulation and display functionality (see Section 5.2.1) that are needed to fulfil the basic functionality of the SDSS (see Section 5.1).

The SDSS for pipe break susceptibility analysis and impact assessment is composed of two basic subsystems, viz. an Object Query Browser and a GIS. The Object Query Browser can be regarded as the control panel of the system from where all the query, manipulation and display functionality can be accessed. The GIS subsystem provides the spatial query, spatial selection and thematic mapping functionality. The GIS subsystem is too large a system to be considered as a subsystem of the Object Query Browser. The Object Query Browser, however, can be seen as a subsystem of the GIS, since it can be incorporated into the GIS using the built-in Visual Basic for Applications (VBA) programming language, which is part of many modern GISs. The Object Query Browser and GIS will, however, be considered as equally important basic subsystems of the SDSS, therefore the *is-composed-of* relationship (see Section 3.1.4.3 and Section 3.2.2) is used to describe the relationship between SDSS and the two basic subsystems (see Figure 5.3). The classes of the Object Query Browser are grouped in packages that follow a slight variation of the classical three-tier architecture (see Section 2.4.1.2), in which the Object Query Browser imports the User_services subsystem, which again imports the Data_services subsystem, which finally imports the Operation_services subsystem (see Figure 5.3). The User_services subsystem thus directly requests the services of the Data_services subsystem. The Operation_services subsystem (which is used instead of the Business Services subsystem, see Section 2.4.1.2) further extends the data query and manipulation services of the Data_services subsystem.

The User_services subsystem is the user interface of the system and can thus be seen as the system boundary (see Section 2.1.3.1), where all inputs enter the system and the appropriate functions and operations to query and manipulate the input data can be selected. The output from the system can be viewed by the various graphical user interfaces (GUIs) that are incorporated into the User_services package. The User_services subsystem uses the *import* relationship (see Section 3.2.6) to share boundaries (see Section 2.1.3.3) with the GIS, the Interpolation_utiltiy, the Hydraulic_library, the Fuzzy_controller, Graph_utility and System_administration subsystems (see Figure 5.3). The interface between the User_services subsystem and these other subsystems will be discussed in Section 5.4.

The User_services subsystem also imports the Data_services subsystem (see Figure 5.3) to maintain, access and update the data in the object-oriented data structure. The classes in the Data_services package are shown in the class diagram of Figure 5.4. An object diagram is given in Figure 5.5, showing a snapshot view of the instances in the system resulting from a query performed on a collection (see Section 3.1.7) of node elements that have associated pipe elements. The object data to be queried in the Object Query Browser, and also the results of these queries, reside in the *queryRes* collection (see Figure 5.4). The *queryRes* collection can be conceptualised as being a collection of various input data collections and query results collections (see Section 3.1.7), presented in a tree-like view for better overview. The items in the *queryRes* collection, i.e. the individual query result collections (or the input data collections to be queried), consist of instances of the Treebranch class (see Figure 5.4). The *ckey* attribute of the Treebranch class is the key to access a specific Treebranch item in the *queryRes* collection. The items of the *cDatarows* collection of the Treebranch class are instances of the TreebranchDatarow class (see Figure 5.4). The *cDatarow* collection of the TreebranchDatarow class contains the attribute values of a single object. The *cDatarows* collection can be interpreted as an attribute table listing all objects in a query results collection, and the *cDatarow* collection can be interpreted as a single data row in the attribute table. The *cproperties* collection of the Treebranch class contains the keys to access the object attribute data in the *cDatarow* collection. The first item in the *cDatarow* collection holds the object identifier (see Section 3.1.1). The object identifier can be used as a key to access the items (i.e. the individual object data rows) in the *cDatarows* collection. Each object in the *cDatarows* collection can have an associated collection containing associated objects. The *cDatacollecs* collection of the Treebranch class contains the associated collections. The items of the *cDatacollecs* collection are instances of the TreebranchDatacollec class (see Figure 5.4). The items of the *cDatacollecRows* collection of the TreebranchDatacollec class consist of instances of the TreebranchDatacollecRow class (see Figure 5.4). The *cDatacollecRow* collection of the TreebranchDatacollecRow class contains the attribute values of a single associated object. The *cDatacollecs* collection can be conceptualised as a file containing all the associated attribute tables of the objects in a query results collection. The *cDatacollecRows* collection represents an associated attribute table and the *cDatacollecRow* collection can be interpreted as single row in the associated attribute table. The above-mentioned object identifier for the objects in the query results collection can also be used as a key to access an object's corresponding associated collection (i.e. the matching associated attribute table) in the *cDatacollecs* collection. The *ccproperties* collection of the Treebranch class contains the keys to access the associated object attribute data in the *cDatacollecRow* collection.

The *queryRes* collection is a private collection of the QueryTree class and the data is thus fully encapsulated (see Section 3.1.5), since it can only be accessed and updated with the QueryTree class operations. The operations in the QueryTree class include file-handling operations (see later Section 5.3.1), navigational operations such as to create new query collections and sub-collections, and operations for establishing the interface between the User_services subsystem of the Object Query Browser and the other subsystems linked to it (see Figure 5.3).

The attribute query functions and attribute manipulation operations, viz. *obj_Check1..10*, *obj_Calc 1..10*, *collec_Calc1..10* and *tree_Calc1..10* could also have been included in the QueryTree class, but have rather been placed in a separate TreeOperations class of the Operation_services subsystem (see Figure 5.6), which is then imported by the Data_services subsystem (see Figure 5.3). In this way object responsibility is distributed more evenly (see Section 3.1.5). The SDSS user can access these functions and operations of the TreeOperations class via the Attribute GUI on different data access levels (see later Section 5.4.3). A series of object interactions will then take place where objects from the Data_services subsystem will be passed on via parameter transfer (i.e. via the dependency relationship, see Section 3.1.4.1) to the operations in the TreeOperations class of the Operation_services subsystem, where queries and manipulations will be carried out on the object data. The encapsulation principle is still adhered to, since the operations in the TreeOperations class can only manipulate the specific objects that they receive as arguments.

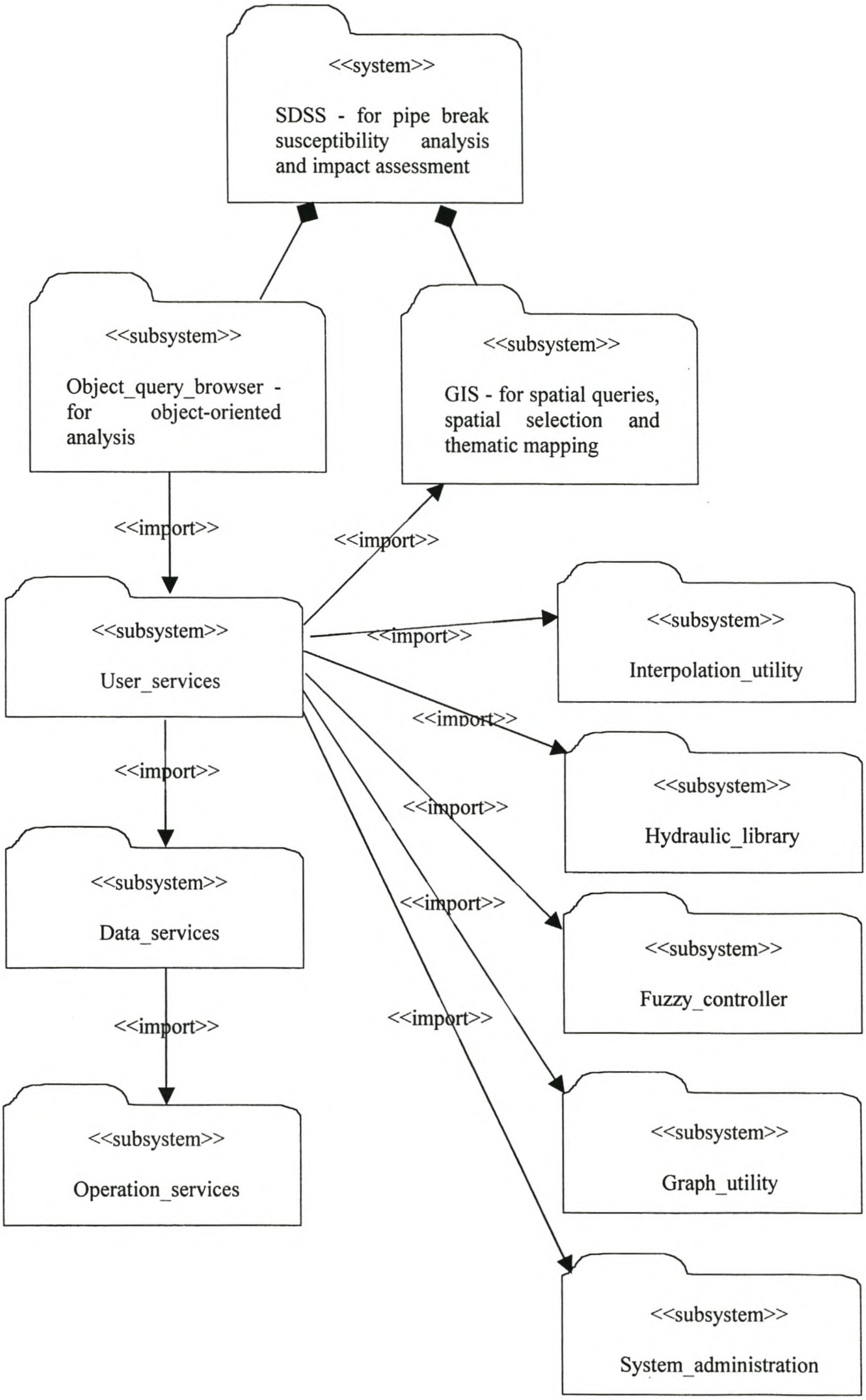


FIGURE 5.3 General design of the SDSS architecture

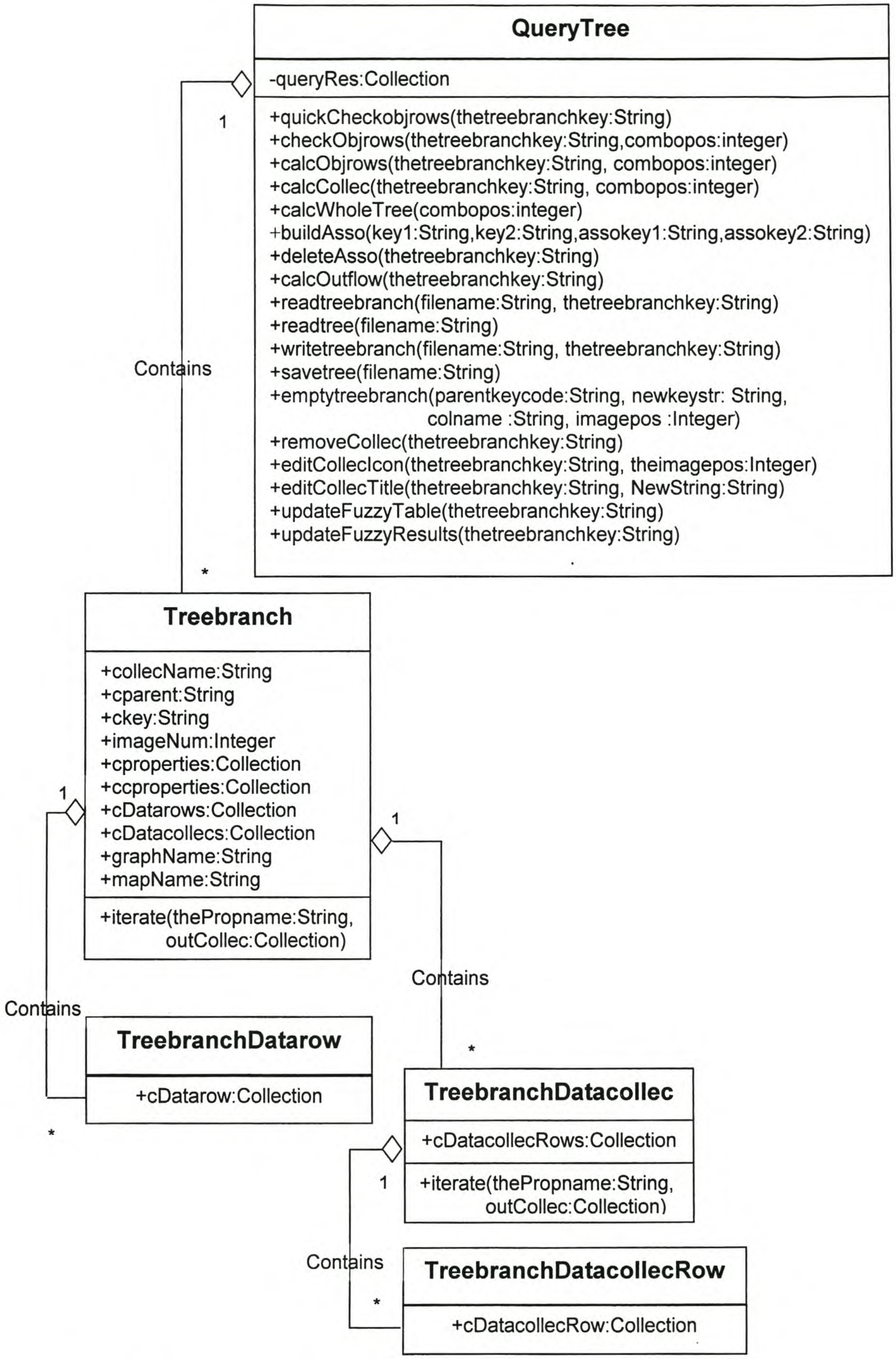

QueryTree
-queryRes:Collection
+quickCheckobjrows(thetreebranchkey:String)
+checkObjrows(thetreebranchkey:String,combopos:integer)
+calcObjrows(thetreebranchkey:String, combopos:integer)
+calcCollec(thetreebranchkey:String, combopos:integer)
+calcWholeTree(combopos:integer)
+buildAsso(key1:String,key2:String,assokey1:String,assokey2:String)
+deleteAsso(thetreebranchkey:String)
+calcOutflow(thetreebranchkey:String)
+readtreebranch(filename:String, thetreebranchkey:String)
+readtree(filename:String)
+writetreebranch(filename:String, thetreebranchkey:String)
+savetree(filename:String)
+emptytreebranch(parentkeycode:String, newkeystr: String, colname :String, imagepos :Integer)
+removeCollec(thetreebranchkey:String)
+editCollecIcon(thetreebranchkey:String, theimagepos:Integer)
+editCollecTitle(thetreebranchkey:String, NewString:String)
+updateFuzzyTable(thetreebranchkey:String)
+updateFuzzyResults(thetreebranchkey:String)
1
Contains
*
Treebranch
+collecName:String
+cparent:String
+ckey:String
+imageNum:Integer
+cproperties:Collection
+ccproperties:Collection
+cDatarows:Collection
+cDatacollecs:Collection
+graphName:String
+mapName:String
+iterate(thePropname:String, outCollec:Collection)
1
Contains
*
TreebranchDatarow
+cDatarow:Collection
1
Contains
*
TreebranchDatacollec
+cDatacollecRows:Collection
+iterate(thePropname:String, outCollec:Collection)
1
Contains
*
TreebranchDatacollecRow
+cDatacollecRow:Collection

FIGURE 5.4 Class diagram: Classes in package Data_services

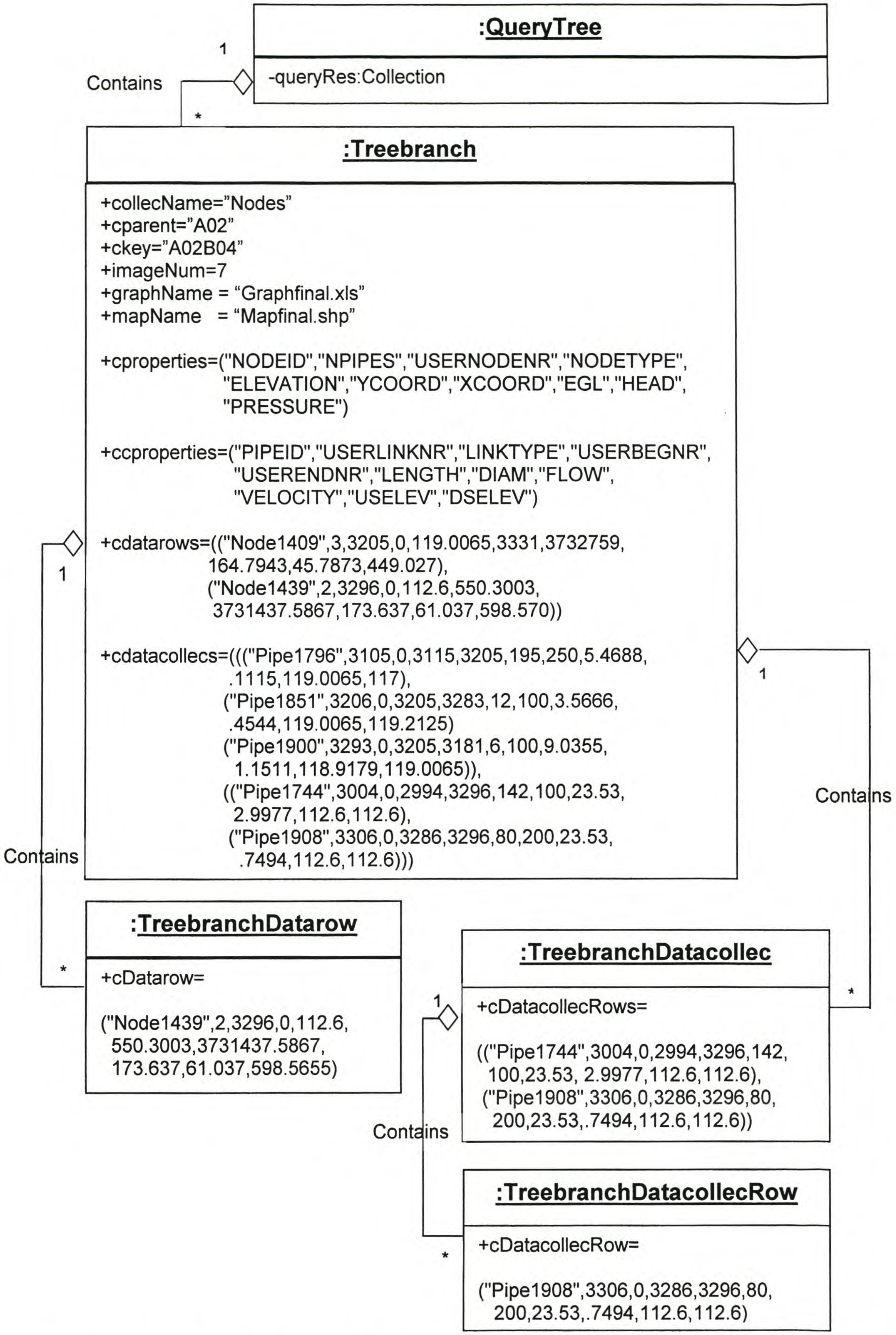
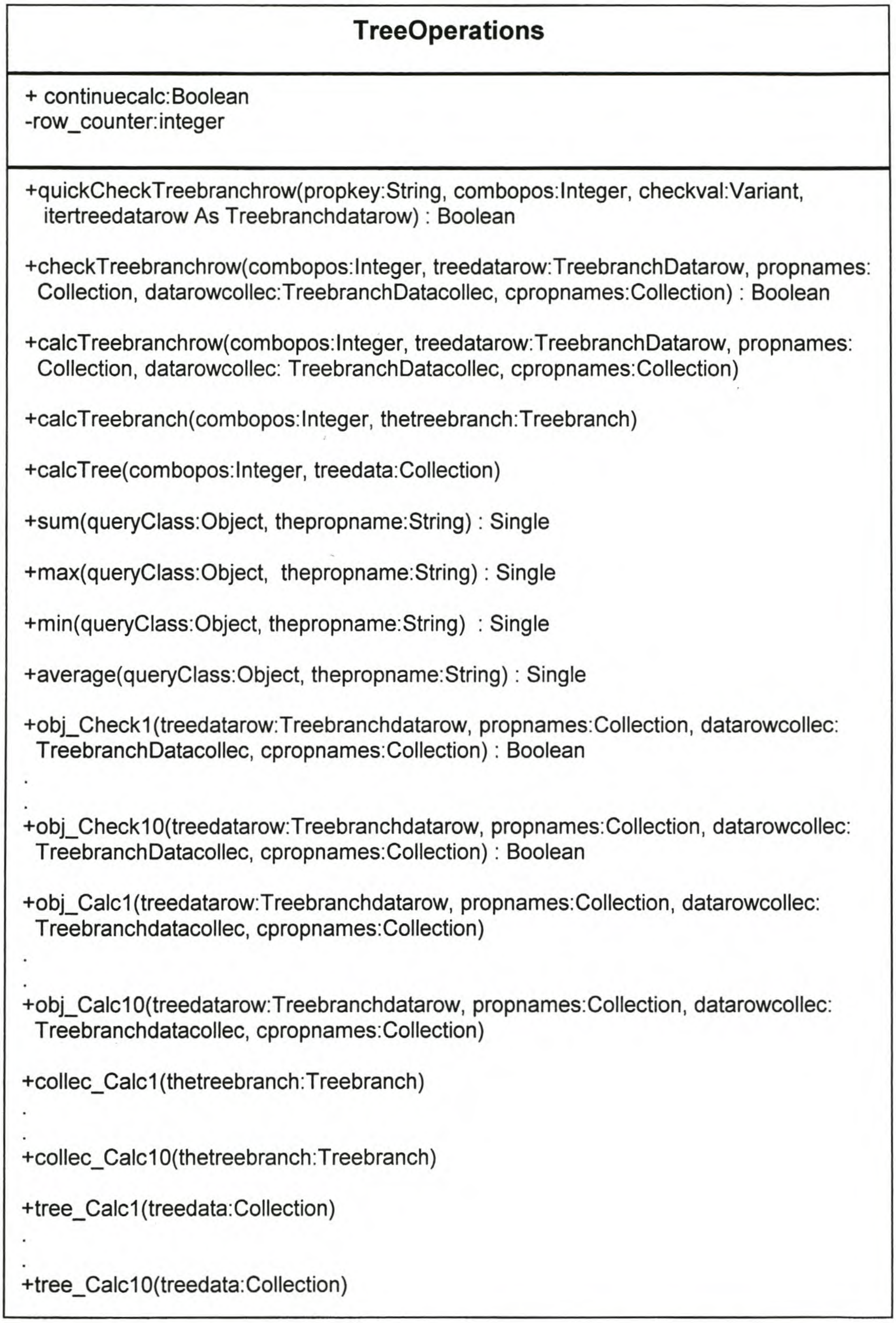

:QueryTree
-queryRes:Collection
Contains
1
*
:Treebranch
+collecName="Nodes"
+cparent="A02"
+ckey="A02B04"
+imageNum=7
+graphName = "Graphfinal.xls"
+mapName = "Mapfinal.shp"
+cproperties=("NODEID","NPIPES","USERNODENR","NODETYPE", "ELEVATION","YCOORD","XCOORD","EGL","HEAD", "PRESSURE")
+ccproperties=("PIPEID","USERLINKNR","LINKTYPE","USERBEGNR", "USERENDNR","LENGTH","DIAM","FLOW", "VELOCITY","USELEV","DSELEV")
+cdatarows=(("Node1409",3,3205,0,119.0065,3331,3732759, 164.7943,45.7873,449.027), ("Node1439",2,3296,0,112.6,550.3003, 3731437.5867,173.637,61.037,598.570))
+cdatacollecs=((("Pipe1796",3105,0,3115,3205,195,250,5.4688, .1115,119.0065,117), ("Pipe1851",3206,0,3205,3283,12,100,3.5666, .4544,119.0065,119.2125) ("Pipe1900",3293,0,3205,3181,6,100,9.0355, 1.1511,118.9179,119.0065)), (("Pipe1744",3004,0,2994,3296,142,100,23.53, 2.9977,112.6,112.6), ("Pipe1908",3306,0,3286,3296,80,200,23.53, .7494,112.6,112.6)))
1
Contains
*
1
Contains
*
:TreebranchDatarow
+cDatarow=
("Node1439",2,3296,0,112.6, 550.3003,3731437.5867, 173.637,61.037,598.5655)
:TreebranchDatacollec
+cDatacollecRows=
(("Pipe1744",3004,0,2994,3296,142, 100,23.53, 2.9977,112.6,112.6), ("Pipe1908",3306,0,3286,3296,80, 200,23.53,.7494,112.6,112.6))
1
Contains
*
:TreebranchDatacollecRow
+cDatacollecRow=
("Pipe1908",3306,0,3286,3296,80, 200,23.53,.7494,112.6,112.6)

FIGURE 5.5 Object diagram: Instances of classes in package Data_services



TreeOperations

+ continuecalc:Boolean
-row_counter:integer

+quickCheckTreebranchrow(propkey:String, combopos:Integer, checkval:Variant, itertreedatarow As Treebranchdatarow) : Boolean

+checkTreebranchrow(combopos:Integer, treedatarow:TreebranchDatarow, propnames: Collection, datarowcollec:TreebranchDatacollec, cpropnames:Collection) : Boolean

+calcTreebranchrow(combopos:Integer, treedatarow:TreebranchDatarow, propnames: Collection, datarowcollec: TreebranchDatacollec, cpropnames:Collection)

+calcTreebranch(combopos:Integer, thetreebranch:Treebranch)

+calcTree(combopos:Integer, treedata:Collection)

+sum(queryClass:Object, thepropname:String) : Single

+max(queryClass:Object, thepropname:String) : Single

+min(queryClass:Object, thepropname:String) : Single

+average(queryClass:Object, thepropname:String) : Single

+obj_Check1(treedatarow:Treebranchdatarow, propnames:Collection, datarowcollec: TreebranchDatacollec, cpropnames:Collection) : Boolean
.
.
+obj_Check10(treedatarow:Treebranchdatarow, propnames:Collection, datarowcollec: TreebranchDatacollec, cpropnames:Collection) : Boolean

+obj_Calc1(treedatarow:Treebranchdatarow, propnames:Collection, datarowcollec: Treebranchdatacollec, cpropnames:Collection)
.
.
+obj_Calc10(treedatarow:Treebranchdatarow, propnames:Collection, datarowcollec: Treebranchdatacollec, cpropnames:Collection)

+collec_Calc1(thetreebranch:Treebranch)
.
.
+collec_Calc10(thetreebranch:Treebranch)

+tree_Calc1(treedata:Collection)
.
.
+tree_Calc10(treedata:Collection)

FIGURE 5.6 Class diagram: Classes in package Operation_services

5.3 DESCRIPTION AND DESIGN OF THE DATA MANIPULATION OPERATIONS IN THE SPATIAL DECISION SUPPORT SYSTEM

This section describes the data manipulation operations of the Object Query Browser that are used in general data analyses and manipulations. In order to facilitate the description of some of the complex data manipulation operations, activity diagrams (see discussion in Section 3.2.5) will accompany the descriptions. The specialised functions and operations for pipe break susceptibility analysis and impact assessment will be discussed later in Chapter 6.

5.3.1 File handling operations

The *writetreebranch* operation of the QueryTree class (see Figure 5.4) makes the current query results collection in the system persistent. The operation writes an item of the *queryRes* collection, which is a Treebranch object containing the current query results, to a text file (see Appendix A). The *readtreebranch* operation of the QueryTree class can read the text file containing the results from a previous query session into a Treebranch object, and then adds it to the *queryRes* collection. The two operations are also called by the copy and paste operations in the Object Query Browser. The copy operation creates - via the *writetreebranch* operation - a temporary internal file which is an exact copy of the selected query results collection. The paste operation can then read in this file via the *readtreebranch* operation. The temporary filename is specified internally by the system and the file is overwritten each time a collection is copied so that hard-disk storage space is not wasted.

The *savetree* operation of the QueryTree class saves to a file all query results collections of the active project (i.e. the entire *queryRes* collection is thus saved). The *readtree* operation, also of the QueryTree class, reads into the system all query results collections from a previously saved project (i.e. previously saved query results are read into the *queryRes* collection). The two operations also write and read respectively a project file containing the query results collection names and corresponding file paths.

5.3.2 Build association

The *buildAsso* operation of the QueryTree class (see Figure 5.4) builds an association between a specified main query results collection and another selected collection (which will be the associated query results collection), based on the comparison between two specified key attributes. The operation can build simple associations only, where the key attribute in the main query results collection is directly compared with the key attribute of the query collection to be associated. The multiplicity of the association will depend on the data structure of both the main and the associated collection. The operation is one of the navigational tools (see later Section 5.4.1) and is implemented in the system with the following signature:

QueryTree::buildAsso (key1:string, key2:string, assokey1:string, assokey2:string)

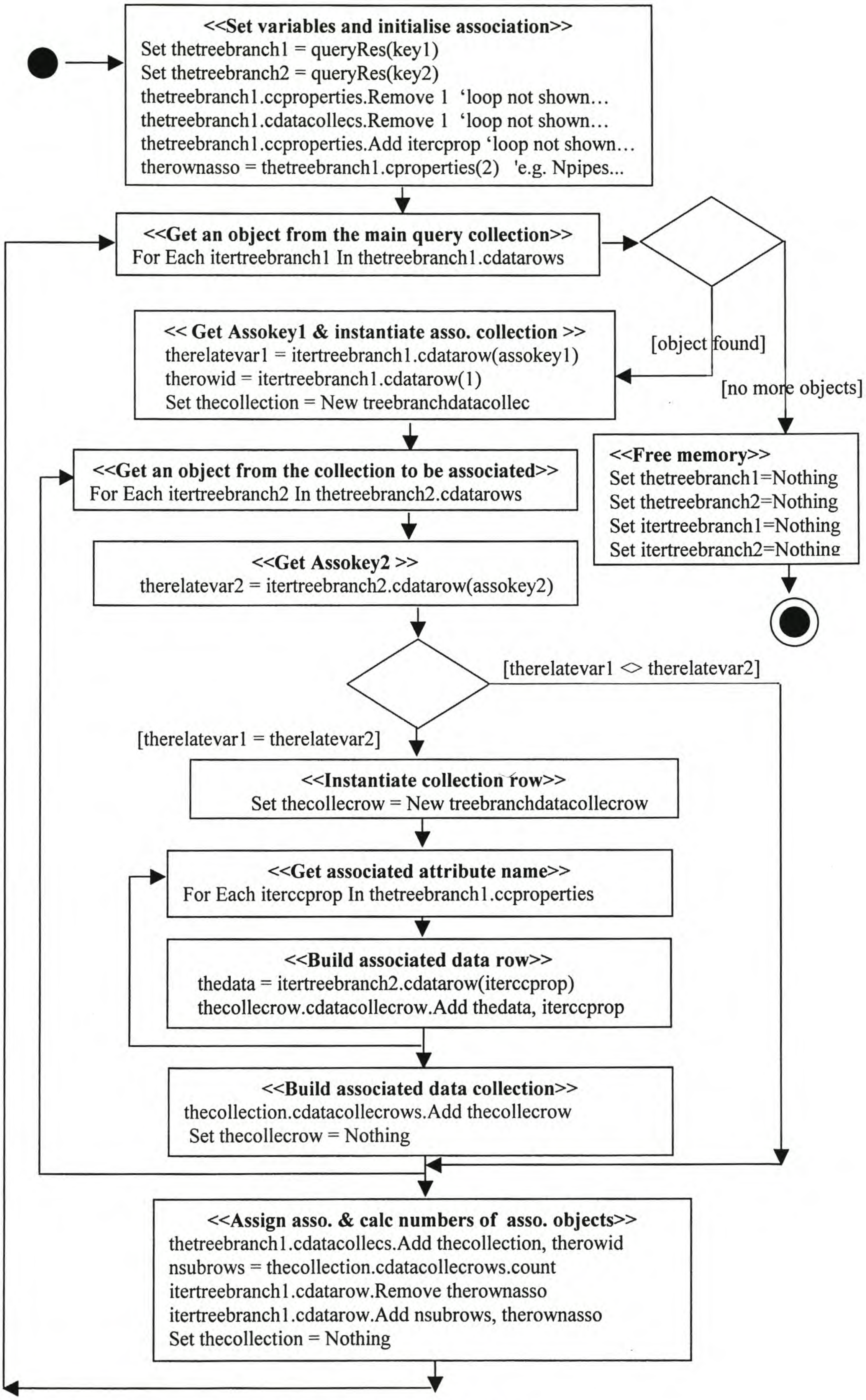
An activity diagram describing the internal flow of the operation is given in Figure 5.7, with the accompanying activity diagram descriptions (see Figure 5.8a,b).

5.3.3 Build complex association

An operation for building complex associations, involving complex association conditions, can be selected in the Object Query Browser (see later Section 5.4.3.3). The operation is implemented in the system as operation *tree_Calc1* of the TreeOperations class (see Figure 5.6) with the following signature:

TreeOperations::tree_Calc1(treedata : Collection)

The operation has an operation flow that is very similar to the *buildAsso* operation. As most other operations and functions of the TreeOperations



```
<<Set variables and initialise association>>
Set thetreebranch1 = queryRes(key1)
Set thetreebranch2 = queryRes(key2)
thetreebranch1.ccproperties.Remove 1 'loop not shown…
thetreebranch1.cdatacollecs.Remove 1 'loop not shown…
thetreebranch1.ccproperties.Add itercprop 'loop not shown…
therownasso = thetreebranch1.cproperties(2) 'e.g. Npipes...

<<Get an object from the main query collection>>
For Each itertreebranch1 In thetreebranch1.cdatarows

<< Get Assokey1 & instantiate asso. collection >>
therelatevar1 = itertreebranch1.cdatarow(assokey1)
therowid = itertreebranch1.cdatarow(1)
Set thecollection = New treebranchdatacollec

<<Get an object from the collection to be associated>>
For Each itertreebranch2 In thetreebranch2.cdatarows

<<Get Assokey2 >>
therelatevar2 = itertreebranch2.cdatarow(assokey2)

<<Instantiate collection row>>
Set thecollecrow = New treebranchdatacollecrow

<<Get associated attribute name>>
For Each iterccprop In thetreebranch1.ccproperties

<<Build associated data row>>
thedata = itertreebranch2.cdatarow(iterccprop)
thecollecrow.cdatacollecrow.Add thedata, iterccprop

<<Build associated data collection>>
thecollection.cdatacollecrows.Add thecollecrow
Set thecollecrow = Nothing

<<Assign asso. & calc numbers of asso. objects>>
thetreebranch1.cdatacollecs.Add thecollection, therowid
nsubrows = thecollection.cdatacollecrows.count
itertreebranch1.cdatarow.Remove therownasso
itertreebranch1.cdatarow.Add nsubrows, therownasso
Set thecollection = Nothing

<<Free memory>>
Set thetreebranch1=Nothing
Set thetreebranch2=Nothing
Set itertreebranch1=Nothing
Set itertreebranch2=Nothing
```

[object found]
[no more objects]
[therelatevar1 <> therelatevar2]
[therelatevar1 = therelatevar2]

FIGURE 5.7 Activity diagram for operation *buildAsso*

Activity description: << **Set variables and initialise association** >>

Sets *thetreebranch1*, which is an instance of the Treebranch class, equal to *queryRes(key1)* that holds the main query results collection.

Sets *thetreebranch2,* which is an instance of the Treebranch class, equal to *queryRes(key2)* that holds the query results of the collection to be associated.

The variables key1, key2, assokey1 and assokey2 are passed on to *buildAsso* via the argument list.

All existing associated attribute names and attribute data are removed with the statements *thetreebranch1.ccproperties.Remove 1* and *thetreebranch1. cdatacollecs.Remove 1.* The statements are within loops that are not shown in the activity diagram, to avoid cluttering the diagram with too much detail.

The associated attribute names for the main collection are set via *thetreebranch1.ccproperties.Add itercprop.* The statement is within a loop (not shown on the diagram) that iterates through the collection to be associated and the attributes in that collection are retrieved and temporarily stored in the iteration variable *itercprop.*

The attribute name to hold for each object in the main collection the number of objects that are associated with it, is read and stored into variable *therownasso*.

Activity description: << **Get an object from the main query collection** >>

The loop *For Each itertreebranch1 In thetreebranch1.cdatarows* iterates through the objects in the main query results collection and retrieves one object at a time to be used in subsequent data processing. When all objects in the main collection have been retrieved, the *buildAsso* operation will terminate and remove temporary collections from memory.

Activity description: << **Get Assokey1 & instantiate asso. collection** >>

The association key of the current object in the main collection is retrieved and stored into the variable *therelatevar1*. The object identifier of the current object in the main collection is retrieved and stored into the variable *therowid.* A new associated collection object, viz. *thecollection*, is instantiated.

Activity description:<<**Get an object from the collection to be associated**>>

The loop *For Each itertreebranch2 In thetreebranch2.cdatarows* iterates through the objects in the collection to be associated and retrieves one object at a time to be used in the subsequent data manipulation.

FIGURE 5.8(a) Activity diagram descriptions for operation *buildAsso (Part I)*

Activity description: << **Get Assokey2** >>

The association key of the current object in the collection to be associated, is retrieved and stored into the variable *therelatevar2.* The association keys of the main collection and the collection to be associated, are then compared. If the*relatevar1* = *thereletavar2* then the association will take place, otherwise the next row in the collection to be associated, will be tested.

Activity description: << **Instantiate collection row** >>

A new associated collection row object, viz. *thecollecrow*, is instantiated.

Activity description: << **Get associated attribute name** >>

The loop *For Each iterccprop In thetreebranch1.ccpoperties* iterates through the associated attributes names.

Activity description:<< **Build associated data row** >>

The associated collection data row, viz. *thecollecrow*, is being built up by adding to it the data values in the current row of the collection to be associated.

Activity description:<< **Build associated data collection** >>

The associated collection, viz. *thecollection*, is being built up by adding to it the calculated associated collection data row, viz. *thecollecrow*.

Activity description: <<**Assign asso. & calc number of asso. objects**>>

The associated collection is then assigned (i.e. finally associated with) to the current object in the main query results collection by the following statement: *thetreebranch1.cdatacollecs.Add thecollection, therowid.* The object identifier, contained in the variable *therowid*, will be used as key to access for a particular object in the main query results collection, its associated collection. The number of objects in the associated collection is calculated and the corresponding attribute in the main query results collection that holds this information, is updated.

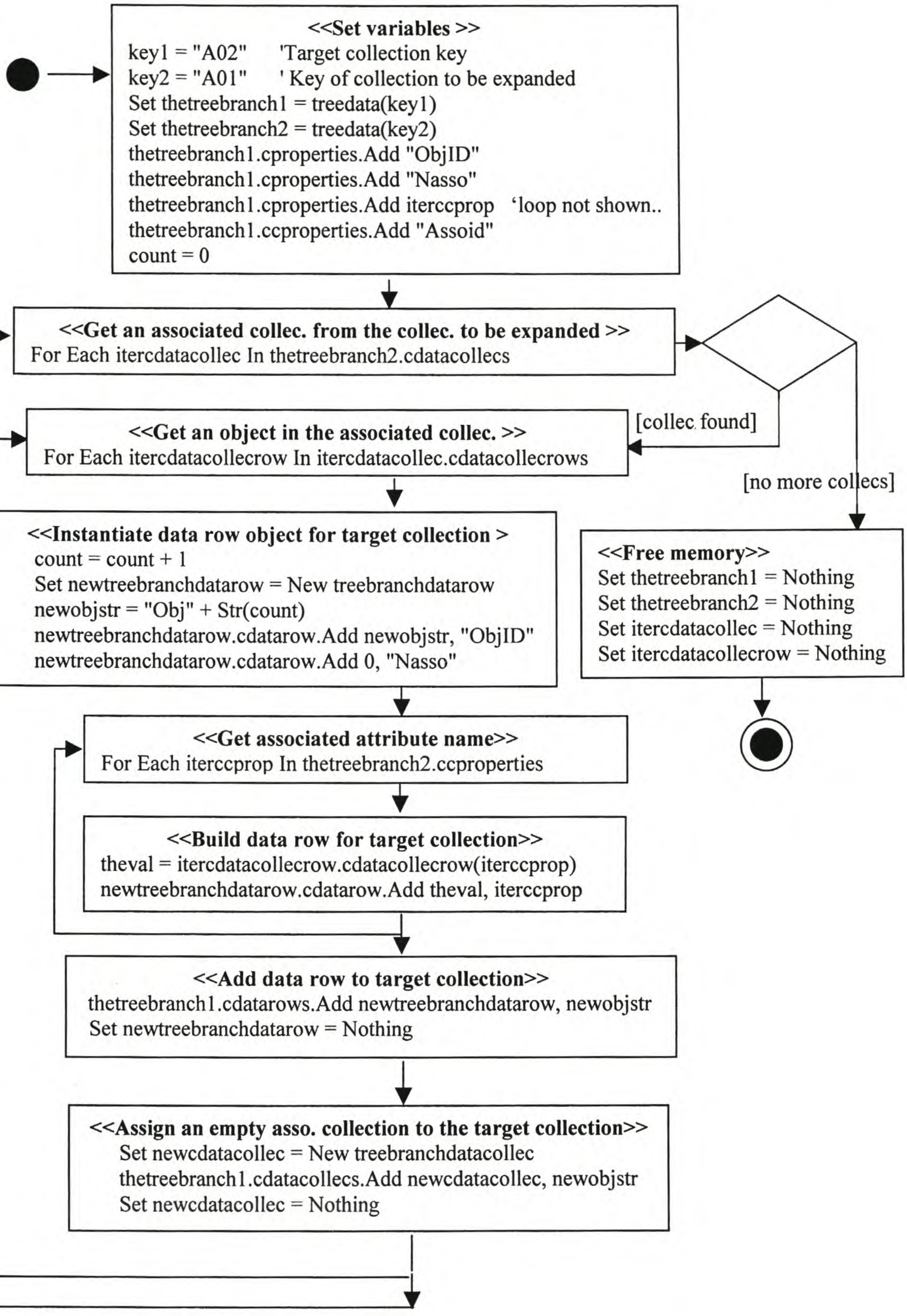
FIGURE 5.8(b) Activity diagram descriptions for operation *buildAsso (Part II)* class, the *tree_Calc1* operation is also editable and can at run-time be interrupted to enter or edit the association condition. An association condition is basically a matching pair function consisting of various attributes that are compared with one another, from both the main query results collection and from the collection to be associated. The multiplicity of the association will depend on the data structure of both the main and the associated collection.

5.3.4 Expand association

An operation to expand the associated collection of a specified query results collection can be selected in the Object Query Browser (see Section 5.4.3.3). The operation will access each object in the associated collection and copy the object over to a new empty collection, which is defined as the target collection. The target collection will then consist of only the associated objects. The operation is implemented in the system as operation *tree_Calc2* of the TreeOperations class (see Figure 5.6) with the following signature:

Treeoperations::tree_Calc2(treedata : Collection)

A new unique object identifier will be generated for each of the associated objects when they are copied into the target collection. The new object identifier will make it possible to distinguish between objects occurring more than once in the target collection, resulting from the expansion of a *many-to-many* association. The original object identifier will, however, for cross-referencing purposes also be stored in the target collection as a normal object attribute. An activity diagram describing the internal flow of the operation is given in Figure 5.9, with the accompanying activity diagram descriptions (see Figure 5.10a,b).



```
<<Set variables >>
key1 = "A02"     'Target collection key
key2 = "A01"     ' Key of collection to be expanded
Set thetreebranch1 = treedata(key1)
Set thetreebranch2 = treedata(key2)
thetreebranch1.cproperties.Add "ObjID"
thetreebranch1.cproperties.Add "Nasso"
thetreebranch1.cproperties.Add iterccprop   'loop not shown..
thetreebranch1.ccproperties.Add "Assoid"
count = 0

<<Get an associated collec. from the collec. to be expanded >>
For Each itercdatacollec In thetreebranch2.cdatacollecs

<<Get an object in the associated collec. >>
For Each itercdatacollecrow In itercdatacollec.cdatacollecrows

<<Instantiate data row object for target collection >
count = count + 1
Set newtreebranchdatarow = New treebranchdatarow
newobjstr = "Obj" + Str(count)
newtreebranchdatarow.cdatarow.Add newobjstr, "ObjID"
newtreebranchdatarow.cdatarow.Add 0, "Nasso"

<<Get associated attribute name>>
For Each iterccprop In thetreebranch2.ccproperties

<<Build data row for target collection>>
theval = itercdatacollecrow.cdatacollecrow(iterccprop)
newtreebranchdatarow.cdatarow.Add theval, iterccprop

<<Add data row to target collection>>
thetreebranch1.cdatarows.Add newtreebranchdatarow, newobjstr
Set newtreebranchdatarow = Nothing

<<Assign an empty asso. collection to the target collection>>
    Set newcdatacollec = New treebranchdatacollec
    thetreebranch1.cdatacollecs.Add newcdatacollec, newobjstr
    Set newcdatacollec = Nothing

<<Free memory>>
Set thetreebranch1 = Nothing
Set thetreebranch2 = Nothing
Set itercdatacollec = Nothing
Set itercdatacollecrow = Nothing
```

FIGURE 5.9 Activity diagram for operation *tree_Calc2* to expand associations

Activity description: << **Set variables** >>

Key1, the target collection key, and *key2* the collection to be expanded, can be specified in an input dialogue box (not shown in the activity diagram).

Sets *thetreebranch1*, which is an instance of the Treebranch class, equal to *treedata(key1)* that holds the empty new target collection.

Sets *thetreebranch2,* which is an instance of the Treebranch class, equal to *treedata(key2)* that holds the query results collection to be expanded.

The first two attribute names are set for the target collection with the following statements: *thetreebranch1.cproperties.Add "ObjID"* and
thetreebranch1.cproperties.Add "Nasso".

The other attribute names for the target collection are set via *thetreebranch1.cproperties.Add iterccprop.* The statement is within a loop (not shown on the diagram) that iterates through the associated collection (*iterccprop* is the iteration variable).

The count variable, used in the automatic generation of the object identifier, is initialised.

Activity description:<<**Get an asso. collec. from the collec. to be expanded**>>

The loop *For Each itercdatacollec In thetreebranch2.cdatacollecs* iterates through the associated collections of the specified query results collection to be expanded and retrieves one associated collection at a time to be used in subsequent data expanding. When all associated collections have been retrieved, the *tree_Calc2* operation will terminate and remove temporary data collections from memory.

Activity description: << **Get an object in the associated collec.** >>

The loop *For Each itercdatacollecrow In itercdatacollec.cdatacollecrows* iterates through the objects in an associated collection and retrieves one object at a time to be expanded.

Activity description:<< **Instantiate data row object for target collection** >>

The new data row object , viz. *newtreebranchdatarow*, is instantiated for the target collection. A new object identifier is also generated and assigned to the data row.

FIGURE 5.10(a) Activity diagram descriptions for operation *tree_Calc2* (Part I)

Activity description:<< **Get associated attribute name** >>

The loop *For Each iterccprop In thetreebranch2.ccproperties* iterates through the associated attribute names of the collection to be expanded.

Activity description:<< **Build data row for target collection** >>

The associated attribute data values of the collection to be expanded, is retrieved and copied to the new data row, viz. *newtreebranchdatarow.*

Activity description:<< **Add data row to target collection** >>

The newly built data row is added to the target collection.

Activity description:<< **Assign an empty asso. collec. to the target collec.** >>

An empty associated data collection, viz. *newcdatacollec*, is instantiated and assigned to the target collection. The target collection will thus have no associated objects.

FIGURE 5.10(b) Activity diagram descriptions for operation *tree_Calc2* (Part II)

5.3.5 Unify collections

An operation for unifying query results collections can be selected in the Object Query Browser (see Section 5.4.3.3). The operation will unify the collections according to the unification method used by the additive-union operator in object algebra (Eaglestone & Ridley 1998). According to this method all objects in the collections to be unified, including those objects that occur more than once, will appear in the final unified collection. The operation can only unify query result collections that have the same type of attributes, which will typically be the case when various query results sub-collections are to be unified again. The operation is implemented in the system as operation *tree_Calc3* of the TreeOperations class (see Figure 5.6) with the following signature:

TreeOperations::tree_Calc3(treedata : Collection)

The operation has an operation flow that is very similar to the *tree_Calc2* operation for expanding associations that was shown in the activity diagram of Figure 5.9. The unify operation, however, copies the objects in the main collection and not the objects in the associated collection to a target collection. New object identifiers are also generated for the objects, just as for the expand operation (see Section 5.3.4), to distinguish between objects occurring more than once in the target collection. An empty target collection can be specified in the beginning as was the case for the expand operation, but it will then gradually grow, since each operation call will copy (append) new objects from the specified query results collection to the target collection to be unified. In this way multiple query results collections can be unified with one another.

5.3.6 Intersect collections

An intersect operation, which is used in object algebra to intersect one collection with the other to obtain the objects that occur in both collections (Eaglestone & Ridley 1998), is not specifically implemented in the Object Query Browser. Intersection between two collections can, however, be accomplished with the Object Query Browser by the following two steps. Firstly, an association between the two collections should be established using *buildAsso* operation (see Section 5.3.2) and the object identifier of the two collections should be specified for the association keys. And, secondly, a query must be specified on the attribute in the main collection holding the number of associated objects, to reduce the main collection so that it contains only objects with one associated object (i.e. Nasso = 1). Hereby all non-matching pairs (i.e. Nasso = 0) will be removed from the collection. Optionally, the collection associated with the reduced main collection can be removed by applying the *deleteAsso* operation of the QueryTree class (see Figure 5.4), which is one of the navigational tools (see Section 5.4.1).

Complex intersection based on a matching pair function, such as can be established with the join operator in object algebra (Eaglestone & Ridley 1998), can also be accomplished by following the above-mentioned steps.

The *tree_Calc1* operation (see Section 5.3.3) will then have to be applied instead of the *buildAsso* operation in order to establish the complex association.

5.3.7 Multifactor Evaluation Process (MFEP)

The Multifactor Evaluation Process (MFEP) is a process that is commonly used to select the best option from a list of alternatives. Before the MFEP can be applied, the decision-maker should first identify the *factors* upon which to judge each alternative. When applying the MFEP, a decision-maker assigns an *importance weight* to each *factor*. The weights can, for example, range from 0 to 1 and they should ideally add up to one. For each alternative, all *factors* are evaluated according to some *factor evaluation* schema that should ideally also range from 0 to 1. The *factor weights* are multiplied by each *factor evaluation* to obtain the *weighted evaluation*, which is then accumulated for each alternative (see Tables 5.1 and 5.2). The alternative with the highest overall score (i.e. the highest accumulated *weighted evaluation* value) is then selected (Render & Stair 1994).

TABLE 5.1 Factor weights

FACTOR NAME	IMPORTANCE (WEIGHT)
Factor_a	0.6
Factor_b	0.3
Factor_c	0.1

TABLE 5.2 Evaluation of factors for an alternative

FACTOR NAME	FACTOR WEIGHTS		FACTOR EVALUATION		WEIGHTED EVALUATION
Factor_a	0.6	x	0.7	=	0.42
Factor_b	0.3	x	0.9	=	0.27
Factor_c	0.1	x	0.6	=	0.06
Total	1				0.75

In site selection studies the MFEP is often the preferred method because of its simplicity, and it is also applied with confidence in decision-making processes involving huge multi-million rand budgets, such as for example in the site selection study for the Mohale Dam of the Lesotho Highlands Water Project (Davies AG 1994).

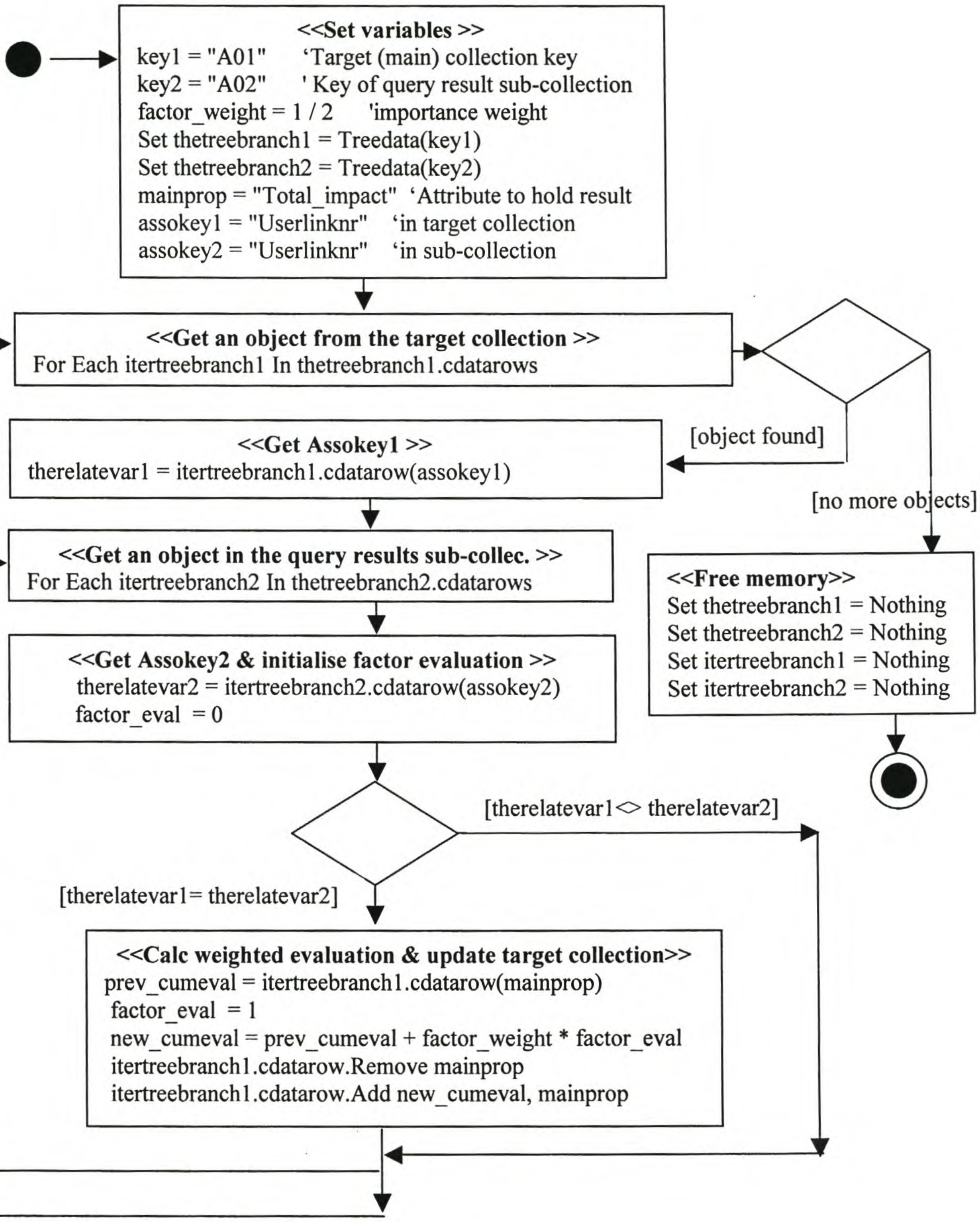
Although the MFEP is mainly used to select the best option from a list of alternatives, it can in very much the same way also be applied in the SDSS to classify objects or phenomena that are based on several independent factors. The MFEP in this case is used to identify and classify all objects in a collection that have an accumulated *weighted evaluation* value of above some specified cut-off point.

The MFEP will be implemented in various sub-systems of the SDSS to summarise and classify the results from various independent sub-analyses (see Chapter 6). These results could also have been summarised and classified using a fuzzy controller (see Section 2.2.3), but this will complicate matters unnecessarily. The fuzzy controller that will be linked to the SDSS (see Section 5.4.4) will be used to model the complex interrelated factors of pipe break susceptibility analysis and impact assessment that cannot be modelled with the simpler MFEP approach.

The MFEP operation can be selected in the Object Query Browser (see Section 5.4.3.3). The operation requires that a target (i.e. the main) query results collection must be specified. The resulting sub-collections of the queries conducted on the target collection must also be specified. The sub-collections can be conceptualised as being *factors* (see Table 5.1 and 5.2) that must be taken into account when classifying the objects in the target collection. The relative importance of the sub-collections, i.e. the *factor weights,* should also be specified. The MFEP operation will evaluate these factors for each object in the target collection. The evaluation schema is very simple, if the factor is applicable to the object in the target collection then a *factor evaluation* value of 1 will be assigned to the target collection object, and if not applicable then a *factor evaluation* value of 0 will be assigned. A *factor* is applicable to the object in the target collection, if that object (i.e. the object in the target collection) is also a member of the sub-collection representing the factor. The MFEP operation will therefore have to search in the sub-collection for the target collection object to determine whether it is a member of the sub-collection. An accumulated *weighted evaluation* value can then be calculated for each object in the target collection, which can then be used in the subsequent object classification. The MFEP operation is implemented in the system as operation *tree_Calc4* of the TreeOperations class (see Figure 5.6) with the following signature:

Treeoperations::tree_Calc4(treedata : Collection)

An activity diagram describing the internal flow of the operation is given in Figure 5.11.



```
<<Set variables >>
key1 = "A01"      'Target (main) collection key
key2 = "A02"      ' Key of query result sub-collection
factor_weight = 1 / 2      'importance weight
Set thetreebranch1 = Treedata(key1)
Set thetreebranch2 = Treedata(key2)
mainprop = "Total_impact"  'Attribute to hold result
assokey1 = "Userlinknr"    'in target collection
assokey2 = "Userlinknr"    'in sub-collection

<<Get an object from the target collection >>
For Each itertreebranch1 In thetreebranch1.cdatarows

<<Get Assokey1 >>
therelatevar1 = itertreebranch1.cdatarow(assokey1)

<<Get an object in the query results sub-collec. >>
For Each itertreebranch2 In thetreebranch2.cdatarows

<<Get Assokey2 & initialise factor evaluation >>
therelatevar2 = itertreebranch2.cdatarow(assokey2)
factor_eval  = 0

<<Calc weighted evaluation & update target collection>>
prev_cumeval = itertreebranch1.cdatarow(mainprop)
factor_eval  = 1
new_cumeval = prev_cumeval + factor_weight * factor_eval
itertreebranch1.cdatarow.Remove mainprop
itertreebranch1.cdatarow.Add new_cumeval, mainprop

<<Free memory>>
Set thetreebranch1 = Nothing
Set thetreebranch2 = Nothing
Set itertreebranch1 = Nothing
Set itertreebranch2 = Nothing
```

[object found]
[no more objects]
[therelatevar1<> therelatevar2]
[therelatevar1= therelatevar2]

FIGURE 5.11 Activity diagram for operation *tree_Calc4* – the MFEP operation

Activity description: << **Set variables** >>

Key1, the target (main) collection key, the query results sub-collection *key2*, and the *factor_weight* indicating the relative importance of the query results sub-collection, can be specified in an input dialogue box (not shown in the activity diagram).

Sets *thetreebranch1*, which is an instance of the Treebranch class, equal to *treedata(key1)* that holds the target collection.

Sets *thetreebranch2,* which is an instance of the Treebranch class, equal to *treedata(key2)* that holds the query results sub-collection.

The attribute *mainprop* of the objects in the target (main) collection to hold the accumulated weighted evaluation results, can be specified in an input dialogue box. *Assokey1* and *assokey2,* for establishing a temporary link between the target collection objects and the query results sub-collection objects, can also be specified in an input dialogue box. A permanent association is, however, not built between the two collections.

Activity description:<< **Get an object from the target collection** >>

The loop *For Each itertreebranch1 In thetreebranch1.cdatarows* iterates through the objects in the target collection and retrieves one object at a time for which the weighted evaluation figure is to be calculated. When all objects in the collection have been retrieved, the *tree_Calc4* operation will terminate and remove temporary data collections from memory.

Activity description: << **Get Assokey1** >>

The association key of the current object in the target (main) collection is retrieved and stored into the variable *therelatevar1*.

Activity description:<< **Get an object in the query results sub-collec.** >>

The loop *For Each itertreebranch2 In thetreebranch2.cdatarows* iterates through the objects in the query results sub-collection.

Activity description:<< **Get Assokey2 & initialise factor evaluation** >>

The association key of the current object in the sub-collection is retrieved and stored into the variable *therelatevar2. The factor_eval* variable is initialised. The association keys of the target collection and the sub-collection are then compared. If the*relatevar1* = *thereletavar2* then the weighted evaluation for the target collection object will be calculated and updated (see next activity description), otherwise the next object (row) in the sub-collection will be tested.

FIGURE 5.12(a) Activity diagram descriptions for operation *tree_Calc4* (Part I)

Activity description:<<**Calc weighted evaluation & update target collection**>>

The previous accumulated weighted evaluation value is retrieved and stored into the variable *prev_cumeval.* The current weighted evaluation value is calculated (viz., *factor_weight* x *factor_eval*) and added to *prev_cumeval* to obtain the new accumulated weighted evaluation value, viz. *new_cumeval.* The accumulated weighted evaluation value for the current object in the target collection is then updated accordingly.

FIGURE 5.12(b) Activity diagram descriptions for operation *tree_Calc4* (Part II)

5.3.8 Statistical functions

There are four built-in statistical functions, viz. *sum*, *max*, *min* and *average*, that can be called from within the *obj_Check* functions and from within the *obj_Calc*, *collec_Calc* and *tree_Calc* operations to calculate the statistics on a specified attribute. The four statistical functions can calculate statistics on attributes of objects in a query results collection or an associated collection. The function will automatically adapt to the type of collection (main or associated collection) via polymorphism (see Section 3.1.6). The operation flow and the object interaction of these statistical functions are rather complex and difficult to describe with activity diagrams and sequence diagrams – the VBA source code, which is not too lengthy, will rather be described directly.

The *sum* function is implemented in the TreeOperations class (see Figure 5.6) of the system with the VBA source code as listed in Figure 5.13. The *queryClass* object in the argument list of the *sum* function (see Figure 5.13) can be an object from the Treebranch class (see Figure 5.4) containing a query results collection (main collection) for which the sum of the attribute specified by the *thepropname* argument, must be calculated. The *queryClass* object can, however, also be an object of the TreebranchDatacollec class (see Figure 5.4) containing an associated collection for which the sum of the attribute specified by the *thepropname* argument must be calculated. The statement *queryClass.iterate thepropname, outCollec* (see Figure 5.13) will call, via late binding (see Section 3.1.6.2), the specific *iterate* operation that is implemented in the class of the *queryClass* object. This implies that when the *queryClass* object is from the Treebranch class then the Treebranch class *iterate* method (see Figure 5.14) will be called. If, however, the *queryClass* object is from the TreebranchDatacollec class then the TreebranchDatacollec class *iterate* method (see Figure 5.15) will automatically be called.

```
Function sum(ByVal queryClass As Object, ByVal
             thepropname As String) As Single

Dim outCollec As New Collection
Dim obj as variant

queryClass.iterate thepropname, outCollec
Sum = 0
For Each obj In outCollec
   Sum = Sum + obj
Next obj
Set outCollec = Nothing

End Function
```

FIGURE 5.13 TreeOperations::sum()

```
Sub iterate(ByVal thepropname As String, OutCollec As
            Collection)
Dim row As New treebranchdatarow

For Each row In cdatarows
  OutCollec.Add (row.cdatarow(thepropname))
Next row
Set row = Nothing

End Sub
```

FIGURE 5.14 Treebranch::iterate()

```
Sub iterate(ByVal thepropname As String, OutCollec As
            Collection)
Dim row As New treebranchdatacollecrow

For Each row In cdatacollecrows
  OutCollec.Add (row.cdatacollecrow(thepropname))
Next row
Set row = Nothing

End Sub
```

FIGURE 5.15 TreebranchDatacollec::iterate()

The specific *iterate* method that has been called will iterate through the collection and will produce the output collection *outCollec* containing the attribute values of the specified attribute *thepropname*. The *sum* function (see Figure 5.13) will then loop through the objects in the *outCollec* collection to calculate the sum figure, which will be the result of the *sum* function. The *sum* function can be called from a *collec_Calc* operation, as shown in Figure 5.16, to calculate the sum of a specified attribute in the active query results collection represented by the object *thetreebranch* (an object from the Treebranch class). The *sum* function can similarly be called from a *tree_Calc* operation to calculate sum statistics on attributes of any query results collection in the project.

```
Sub collec_Calc1(thetreebranch As treebranch)
Dim thepropname As String

thepropname = "Final_result"
thesum = Sum(thetreebranch, thepropname)
.
.
End Sub
```

FIGURE 5.16 Calling the *sum* function from within a *collec_Calc* operation

The *sum* function can also be called from an *obj_Calc* operation, as shown in Figure 5.17, to calculate the sum of a specified attribute in a collection that is associated with the current object in the active query results collection. The associated collection is represented by the object *datarowcollec* (an object from the TreebranchDatacollec class). Iteration through the active collection is not shown in Figure 5.17, since for all *obj_Calc* operations the iteration is managed by the operation *calcObjrows* (see Section 5.4.3.1).

```
Sub obj_Calc4(treedatarow As treebranchdatarow, propnames
  As Collection, datarowcollec As TreebranchDatacollec,
  cpropnames As  Collection)

thesubprop = "Singlerisk"        'Associated property
thesum = Sum(datarowcollec, thesubprop)
.
.
End Sub
```

FIGURE 5.17 Calling the *sum* function from within an *obj_Calc* operation

The other three statistical functions *max*, *min* and *average* have signatures and object interactions very similar to the *sum* function. The *max*, *min* and *average* function calls can be added after the *sum* function call in the *collec_Calc1* operation (see Figure 5.16). The *collec_Calc1* operation can then be selected in the Object Query Browser to generate a statistical report on a specified attribute of the objects in the active query result collection (see Section 5.4.3.2).

5.3.9 Model calibration statistics

The SDSS uses complex models to identify and classify pipes with high break susceptibility (see Chapter 6). If there is data available on pipe break occurrence regarding the location and break causes (see Section 7.1.3), then this data can be used to test and calibrate the pipe break susceptibility analysis model that is used in the SDSS.

The pipe break occurrence data can be imported into the Object Query Browser of the SDSS and then be grouped into separate collections according to the recorded pipe break causes. The query result collections from the pipe break susceptibility analysis can be associated with the pipe break occurrence collection by applying the *buildAsso* operation in Object Query Browser (see Section 5.3.2). An attribute existent in both collections, such as the user link number of the pipe, can be used as an association key to establish the *one-to-many* association (see Section 3.1.4.3) between each pipe and its associated breaks. At first the pipe break susceptibility results and occurrence data should be associated for single pipe break causes only, i.e. the testing and calibration should at first be done for a single pipe break cause, before attempting to test and calibrate multiple pipe break causes.

When testing or calibrating the pipe break susceptibility model, the model calibration statistics consisting of the following three factors should be evaluated collectively:

(i) The *sharpness factor* is a measure to evaluate the size of the query results collection in relation to the total number of pipes in the network. The smaller the factor, the 'sharper' (or 'finer') is the query results collection. The sharpness factor is defined by the following:

$$\frac{\text{Number of pipes in query results collection}}{\text{Total number of pipes in network}} \bullet \frac{100}{1} \qquad (5.1)$$

(ii) The *success rate factor* indicates how many breaks of the actual pipe break occurrences the model was able to predict. The larger the factor is, the better the model prediction. The success rate factor is defined by the following:

$$\frac{\sum \text{nAsso}}{\text{Total number of pipe break occurrences}} \bullet \frac{100}{1} \qquad (5.2)$$

Where, ∑nAsso is the total number of associated pipe break occurrences for the query results collection, i.e. the sum of the *Nasso* attribute for each object in the query results collection.

(iii) The *reliability factor* indicates how many pipes in the active query results collection have in fact broken. The larger the factor is, the better the reliability. The reliability factor is a control measure to check if the model is defined correctly. In the *extremely high* pipe break susceptibility categories, the reliability factor should be almost 100%, confirming that almost all pipes identified by the SDSS that are in these extremely high categories have indeed broken. The reliability factor is defined by the following:

$$\frac{\sum \text{nAsso}}{\text{Number of pipes in query results collection}} \bullet \frac{100}{1} \qquad (5.3)$$

Where, ∑nAsso is defined as in equation 5.2

The testing or calibration period should not be chosen to be too long (one to three years is the recommended period), since there will then be so many recorded pipe break occurrences in the network that the *reliability factor* will be unrealistically large.

An operation to generate the calibration statistics of the pipe break susceptibility analysis model can be selected in the Object Query Browser (see Section 5.4.3.2). The operation is implemented in the system as operation *collec_Calc2* of the TreeOperations class (see Figure 5.6) with the following signature:

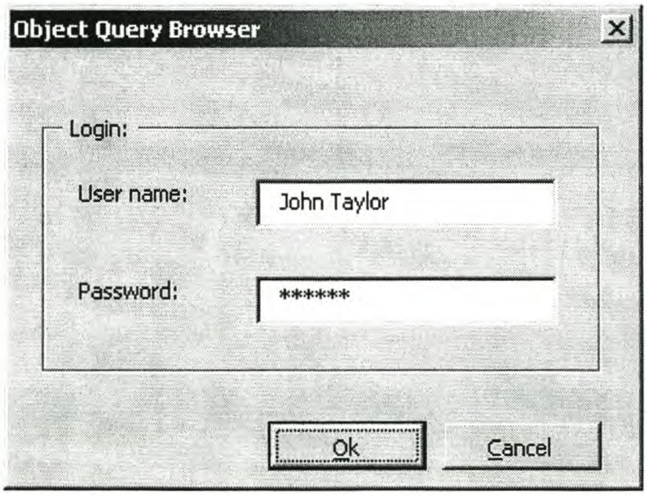
Treeoperations::collec_Calc2(thetreebranch : Treebranch)

The total number of pipes in the network and the total number of actual pipe break occurrences (of a single cause or multiple cause, depending on the calibration level) should be entered. The operation will then calculate the factors according to equations 5.1, 5.2 and 5.3 and present the results in a list box. The *sum* function (see Section 5.3.8) is applied in the operation to determine ∑nAsso that is used in the calculation of the *success rate factor* and the *reliability factor* (see equations 5.2 and 5.3). It will not be necessary to show an activity diagram for this operation, since the activity flow is self-explanatory.

The calibration of the pipe break susceptibility analysis model will further be discussed in Section 7.2 when the SDSS will be applied to the water distribution system of a study area. The pipe break impact assessment model that is used in the SDSS can also be tested and calibrated in very much the same way as described for the pipe break susceptibility analysis model, provided that there are data available on the topography and the extent of the damage caused by the pipe breaks on surrounding properties.

5.4 DESCRIPTION AND DESIGN OF THE USER INTERFACE OF THE SPATIAL DECISION SUPPORT SYSTEM

The Object Query Browser (see Section 5.2.2) is the central control panel of the SDSS from where all the functionality within the SDSS can be accessed. The Object Query Browser, which is written in Visual Basic for Applications (VBA), can easily be fully integrated in any modern GIS that has a built-in VBA environment (see Section 5.2.2), such as for example ArcView 8.1. The Object Query Browser has seven graphical user interfaces (GUIs) where the user can have access to the following types of functionality: (i) Navigation – for overview and grouping of query result collections; (ii) Spatial – to specify spatial queries (and selections), to interpolate elevation points and to apply hydraulic functions for determining the outflow from a pipe break; (iii) Attribute – for non-spatial attribute queries and manipulations; (iv) Fuzzy – for fuzzy-logic-based analysis; (v) Graphs – to display the results using graphs; (vi) Thematic – for thematic mapping of the results; and (vii) Administration – to update user access rights. Not all GUIs will be accessible to the different users of the system. Every time the Object Query Browser starts up a login box will appear (see Figure 5.18) where the user name and corresponding password must be entered. The system will then adapt the GUIs accordingly, i.e. disable (dim out) the GUIs that are not accessible to the specific type of user or deny access if the specified user name and corresponding password combination is invalid (see Section 5.4.7).



User name: John Taylor

Password: ******

FIGURE 5.18 Object Query Browser: Login box

The seven GUIs of the Object Query Browser will be discussed in the sections that follow.

5.4.1 Navigation Graphical User Interface (GUI)

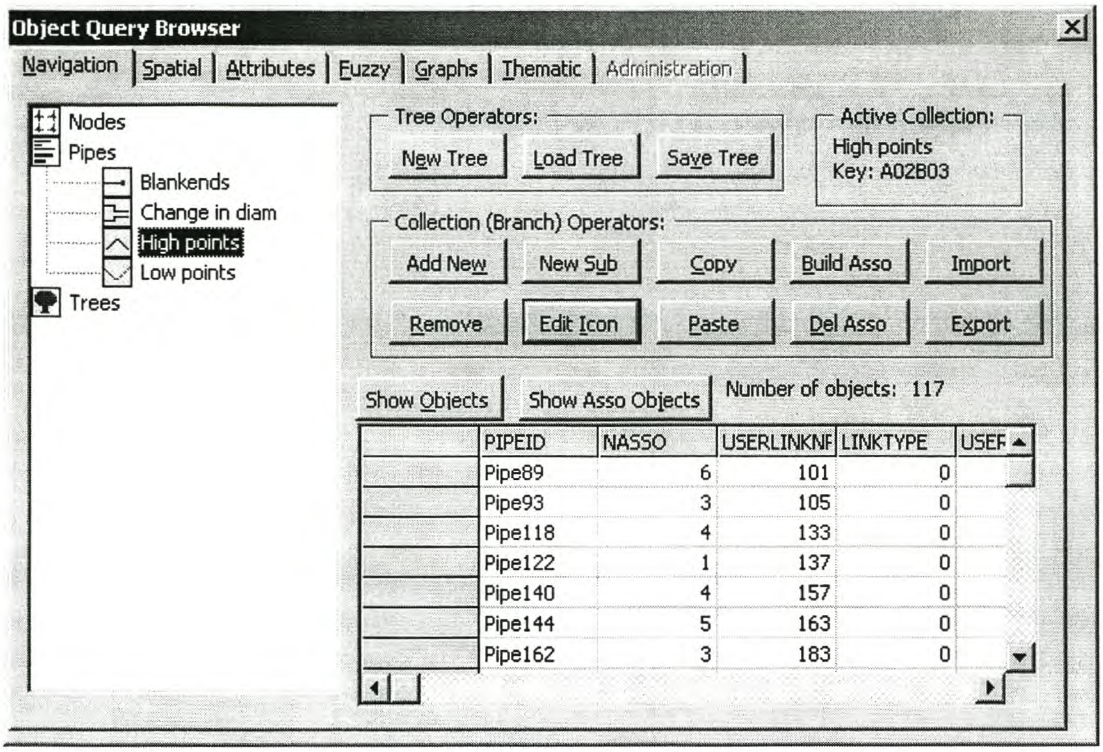
The Navigation GUI (see Figure 5.19) is the main GUI of the Object Query Browser with which the query results collections (simply referred to as query collections) can be grouped in a tree-like structure for a better overview. In the Navigation GUI new query collections can be added or removed, copied and then pasted, associated with other collections, imported or exported, and the objects in the collection (as well as in the associated collection) can be viewed. The Navigation GUI (see Figure 5.19) can be described as follows:

- The *New Tree* button removes all query collections in the tree-like browser, i.e. it initialises the Object Query Browser.

- The *Load Tree* button invokes the *readtree* operation (see Section 5.3.1) to load into the browser an existing query collection tree. It hereby opens an existing project.

- The *Save Tree* button invokes the *savetree* operation (see Section 5.3.1) to save the current query collection tree. It hereby saves the current project.

- The *Add New* button can be clicked to open a new query collection in the tree-like browser directly below the active query collection and with the same indentation. The active query collection is the currently selected query collection and is indicated in the *Active Collection* box in the top right corner of the Navigation GUI. When the *Add New* button is clicked, a new collection identification key will also automatically be generated and assigned to the new collection. If the active query collection has, for example, the key A01, then the new query collection will be added to the tree directly below it with the generated key A02.



Active Collection:
High points
Key: A02B03

Number of objects: 117

	PIPEID	NASSO	USERLINKNF	LINKTYPE	USEF
	Pipe89	6	101	0	
	Pipe93	3	105	0	
	Pipe118	4	133	0	
	Pipe122	1	137	0	
	Pipe140	4	157	0	
	Pipe144	5	163	0	
	Pipe162	3	183	0	

FIGURE 5.19 Object Query Browser: Navigation GUI

- The *Remove* button will remove the active query collection from the tree.

- The *New Sub* button can be clicked to open a new query collection in the tree-like browser directly below the active query collection, but with increased indentation. The newly created collection can be viewed as a sub-collection of the active query results collection in which the more refined query results can be stored. A new collection identification key will also automatically be generated and assigned to the new sub-collection. If the active query collection has, for example, the key A01, then the new query sub-collection will be added to the tree directly below it, but indented to the right of it with the generated key A01B01.

- The *Copy* button will copy the contents of the active query collection into a temporary buffer (see Section 5.3.1).

- The *Paste* button will paste the contents of the temporary buffer to the currently selected location in the tree. With *Copy* and *Paste* exact copies of query collections can be made (see Section 5.3.1).

- The *Build Asso* button can be clicked to call the *buildAsso* operation to build a simple association (see Section 5.3.2) between the objects in the active query collection with the objects in another specified query collection. When the *Build Asso* button is clicked, a dialogue box will appear where the identification key of the collection with which the active query collection is to be associated with, can be specified as well as the attributes to be used as association keys (see Section 5.3.2). If complex associations are to be built involving complicated association conditions, then the *tree_Calc1* operation (see Section 5.3.3) can be called from the Attributes GUI (see Section 5.4.3.3).

- The *Del Asso* button deletes associations, i.e. it removes the collection associated with the active query collection.

- The *Export* button invokes the *writetreebranch* operation (see Section 5.3.1) to export the active collection to a text file (see Appendix A). The active query collection is made persistent in this way.

- The *Import* button invokes the *readtreebranch* operation (see Section 5.3.1) to import a collection that was previously exported.

- The *Show Objects* button displays in a table the attribute values of the objects in the active query collection.

- The *Show Asso Objects* button displays the attribute values of the objects in the collection that is associated with the active query collection.

- The *Edit Icon* button opens the Collection Icon Designer window (see Figure 5.20). The window contains a list of available icons from which a suitable icon can be selected for the current query collection in the Navigation GUI. The query collection can be identified more easily in this way, especially when there are many query collections listed in the tree view. With the Collection Icon Designer, new icons (in bitmap file format) can be loaded in and added to the existing list of icons, new icons can be created, or existing icons can be edited. A description of the Collection Icon Designer follows:

  ⇒ New icons (in bitmap file format) can be loaded into the Collection Icon Designer by clicking the *Load in new Icon* button (see Figure 5.20) and selecting the file from the standard windows file dialogue box that will appear. The new icon will then be added to the list.

  ⇒ For creating new icons the *Create new Icon* button can be clicked to execute the MS Paint program (see Figure 5.21), which is part of the standard Microsoft Windows accessories. The file *blank.bmp*, which is a blank icon template, can be included in the command line call so that when the MS Paint program starts up, the blank icon template appears where the new icon can be drawn using the standard MS Paint drawing tools. Once the new icon has been created, a suitable filename should be specified (other than *blank.bmp*, so that the template file is not overwritten) when saving the icon. The newly created icon can then finally be loaded into the Collection Icon Designer by clicking the *Load in new Icon* button as described earlier.

  ⇒ The currently selected icon in the Collection Icon Designer window can be edited by clicking the *Edit Icon* button. The MS Paint program will then start up and automatically load in the selected icon bitmap file, since the selected bitmap filename has been included in the command line call. Once the editing and saving the icon in MS Paint have been completed, the *Update Edits* button in the Collection Icon Designer window will become active. The *Update Edits* button can then be clicked to reload the edited icon bitmap file into the Collection Icon Designer.

⇒ The *Assign Icon* button can be clicked to assign the selected icon to the currently active query collection in the Navigation GUI.

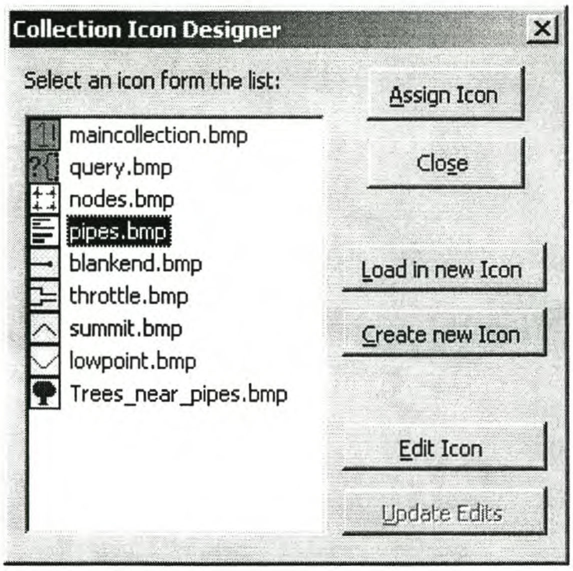
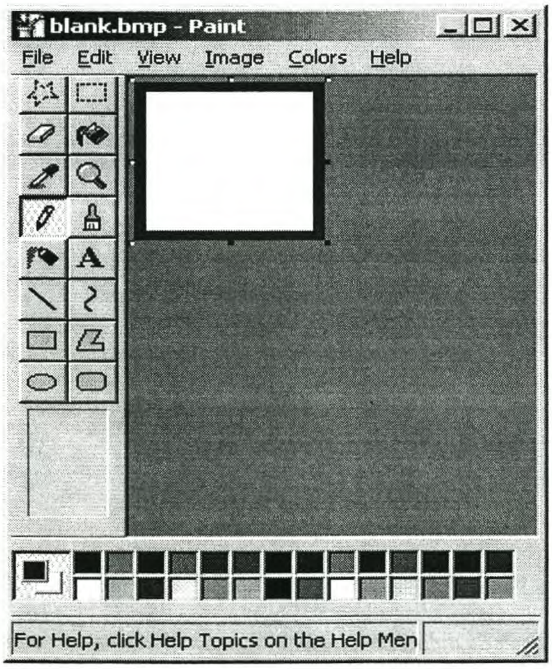


FIGURE 5.20 Navigation GUI: Collection Icon Designer



For Help, click Help Topics on the Help Men

FIGURE 5.21 Collection Icon Designer: MS Paint

5.4.2 Spatial Graphical User Interface (GUI)

The Spatial GUI of the Object Query Browser consists of three sub-GUIs, viz. Spatial Analysis, Topographical Interpolation and Pipe Break Outflow (see Figure 5.22). Each sub-GUI will be described under the sections that follow.

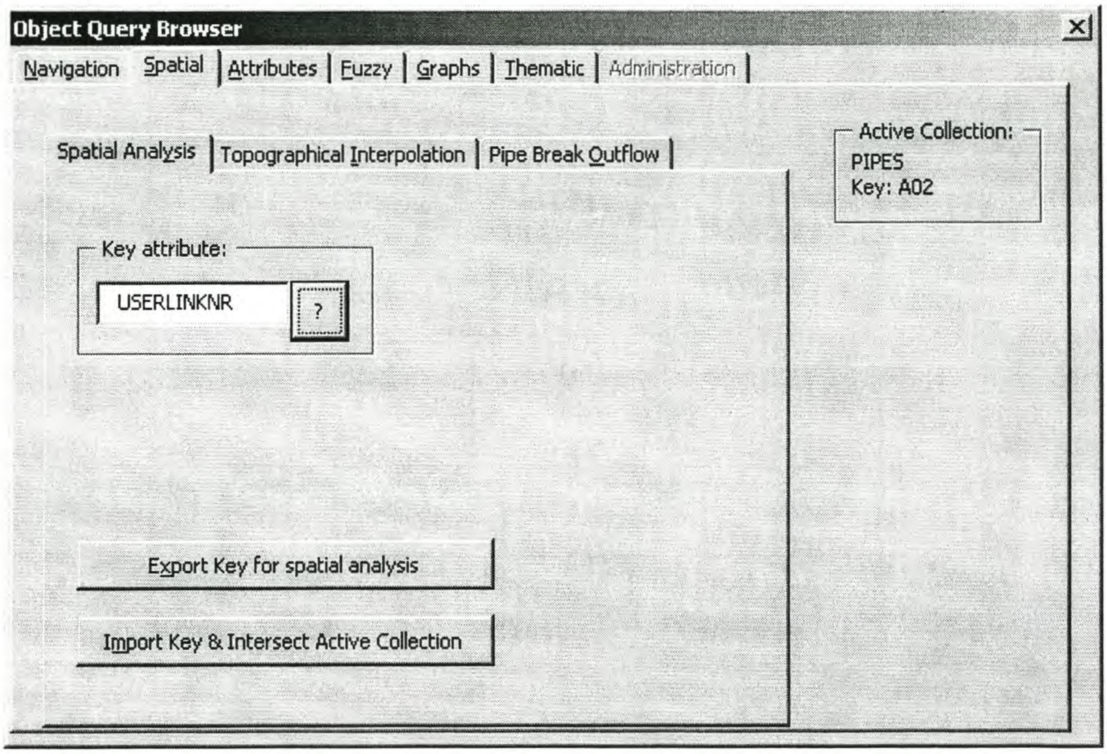
5.4.2.1 Spatial analysis

Spatial analysis functionality, such as spatial query and spatial selection functionality, can be accessed from the Spatial Analysis sub-GUI of the Spatial GUI (see Figure 5.22). The Spatial Analysis sub-GUI links the SDSS with ArcView GIS, where the spatial analysis can be carried out. ArcView GIS, developed by the Environmental Systems Research Institute (ESRI), is a sophisticated desktop mapping application with which spatial data can be visualised, explored, queried and analysed geographically (Hutchinson & Daniel 1995; ESRI 1996; ESRI 2001). The SDSS is temporarily linked with ArcView version 3a, since an academic version of 8.1 (the first version with the built-in VBA) was not available at the time of developing this SDSS. In ArcView version 8.1 a seamless link can be established between the two applications, since the Object Query Browser VBA application can then run directly from within the ArcView 8.1 environment as a VBA macro. A method to link the spatial and attribute data will now be described (and can be used in ArcView version 8.1 as well).

The spatial geometry of the objects in the Objects Query Browser are stored and managed in ArcView shapefiles. The accompanying ArcView attribute tables of these shapefiles will have only one column for the spatial entity identifier. The spatial entities can be linked with the attributes in the Object Query Browser by using the spatial entity identifier as association key. No other attributes are stored in the ArcView attribute tables, since this can lead to data duplication. Attributes are stored centrally in the Object Query Browser where they can be accessed via the Attributes GUI (see Section 5.4.3). A description of the Spatial Analysis sub-GUI (see Figure 5.22) will now follow:

- The key attribute for linking the object attributes in the active query collection with the spatial entities in ArcView can be selected from the *Key attribute* box.

- The *Export Key for spatial analysis* button can be clicked to export (i.e. transfer) the key attribute data to ArcView. ArcView will then also start up, if it is not already active.



Key attribute: USERLINKNR

Active Collection:
PIPES
Key: A02

FIGURE 5.22 Object Query Browser: Spatial GUI & Spatial Analysis sub-GUI

- The spatial theme should be selected in ArcView on which the spatial analysis will be carried out. The first of the four customised ArcView buttons that are located in the top right corner above the scale box, with the icon (see Figure 5.23), can then be clicked to join in the key data. The corresponding spatial entities will then automatically be selected and will appear highlighted in the ArcView graphical view. Further spatial analyses, such as theme on theme selections, etc., can then be accomplished by applying the standard ArcView spatial tools.

- The results from the spatial analysis can be transferred to the Object Query Browser, by clicking the second customised ArcView button with the icon ABC→ (see Figure 5.23). A dialogue box will appear to enable the user to specify whether only the key attributes of the entities in the final spatial selection/query should be transferred to the Object Query Browser or whether the whole attribute table should be transferred. The latter will be the case when a spatial join operation is conducted in ArcView, which automatically adds (and updates) a new temporary *distance* field and a new temporary key attribute field to the ArcView attribute table. This updated ArcView attribute table is then exported into the query result file format (see Appendix A), which can then be imported into the Object Query Browser by applying the *Import* navigational button (see Section 5.4.1). The temporary fields that have been created in the ArcView attribute table can then be dropped to avoid unnecessary data duplication. The first option in the dialogue box must be selected when only the key attributes of the entities in the final spatial selection/query should be transferred to the Object Query Browser. The *Import Key & Intersect Active Collection* button (see Figure 5.22) can then be clicked to update the active query collection, i.e. to intersect the active query collection in the Object Query Browser with the spatial query results obtained from ArcView.

A spatial analysis can also be conducted in ArcView without any query results from the Object Query Browser as starting point for the spatial queries/selection. In this case the *Export Key for spatial analysis* button is not applicable. The query results can then, when the spatial analysis has been completed, be transferred to the Object Query Browser as described earlier.

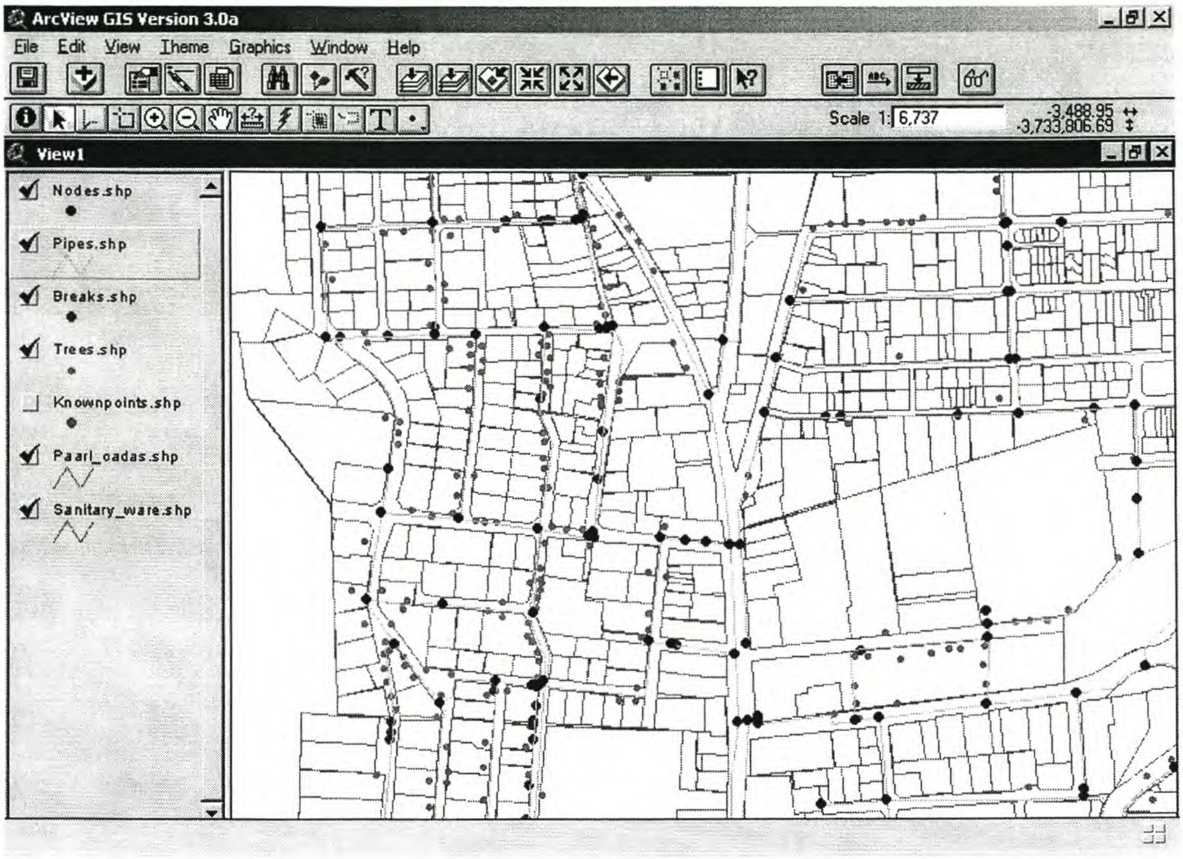


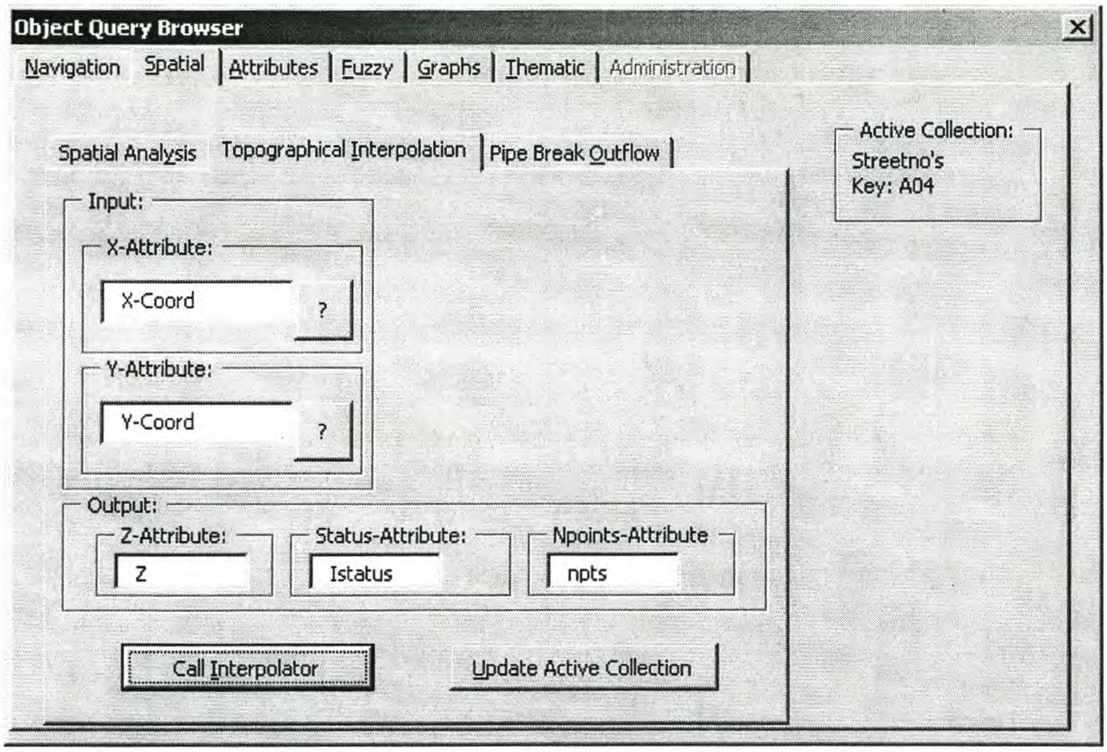
FIGURE 5.23 Spatial analysis in ArcView

5.4.2.2 Topographical interpolation

Topographical interpolation functionality for interpolating the elevation at certain points of unknown elevation, which are surrounded by points of known elevation, can be accessed from the Topographical Interpolation sub-GUI of the Spatial GUI (see Figure 5.24). The Topographical Interpolation sub-GUI links the SDSS with WADPOL, a stand-alone interpolation program that is used in the WADISO S.A. water distribution system program as an accessory for the interpolation of nodal elevations. A description of the Topographical Interpolation sub-GUI (see Figure 5.24) will now follow:

- The attributes in the active query collection containing the X and Y coordinates of the point objects for which the elevations are to be interpolated must be specified in the *Input* box.

- The names of the attributes that will contain the interpolation results must be entered in the respective slots of the *Output* box. The *Z-Attribute* will contain the interpolated elevation. The *Status-Attribute* will contain the interpolation status, i.e. whether interpolation was possible or only extrapolation was possible. The number of known elevation points found in the specified search block surrounding the point for which an elevation is to be interpolated, will be stored in the *Npoints-Attribute*.



X-Attribute: X-Coord

Y-Attribute: Y-Coord

Z-Attribute: Z

Status-Attribute: Istatus

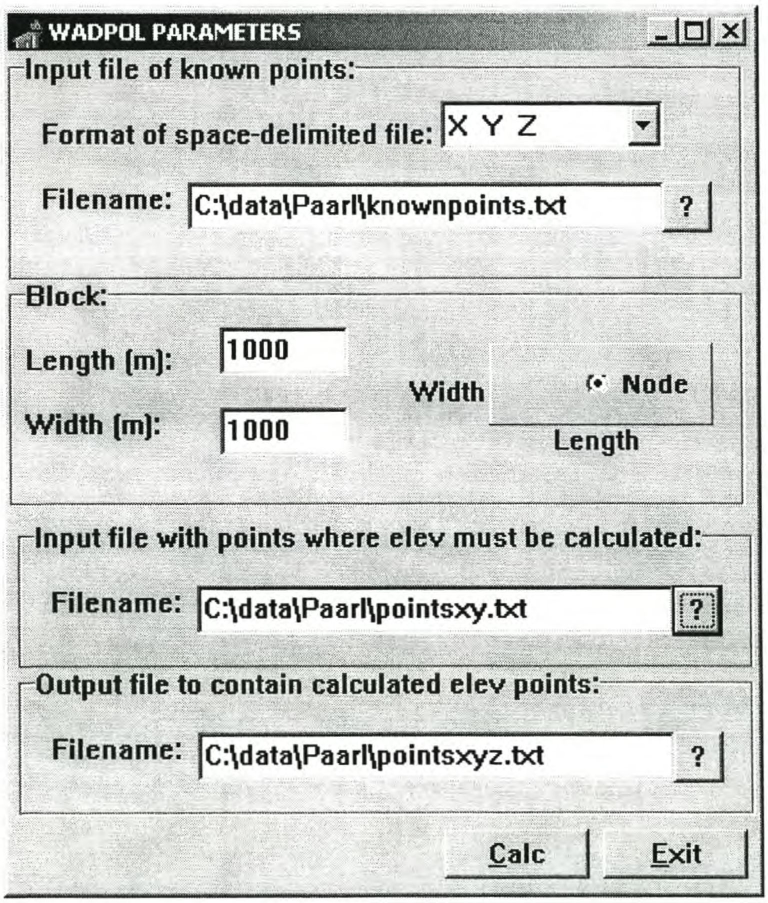
Npoints-Attribute npts

Active Collection:
Streetno's
Key: A04

FIGURE 5.24 Spatial GUI: Topographical Interpolation sub-GUI

- The *Call Interpolator* button can be clicked to call the WADPOL interpolation program, where interpolation parameters can be specified in the WADPOL Parameters window (see Figure 5.25) before starting the interpolation process. The input file containing the known elevations points in the study area must be specified and the file format can be selected from the list of supported file formats. The interpolation search block dimensions must be specified. At each point where an elevation is to be interpolated, the WADPOL interpolator will search for surrounding known elevation points to be used in the interpolation that are within the specified search block extent. The input file, containing the X and Y coordinates of the points for which the elevations are to be interpolated, is automatically created when the *Call Interpolator* button is clicked in the Topographical Interpolation sub-GUI.

- When the WADPOL interpolation process is completed, an output file containing the interpolation results is created. The *Update Active Collection* button in the Topographical Interpolation sub-GUI can then be clicked to update the active query collection with the interpolation output results.



Input file of known points:

Format of space-delimited file: X Y Z

Filename: C:\data\Paarl\knownpoints.txt

Block:

Length (m): 1000

Width (m): 1000

● Node

Input file with points where elev must be calculated:

Filename: C:\data\Paarl\pointsxy.txt

Output file to contain calculated elev points:

Filename: C:\data\Paarl\pointsxyz.txt

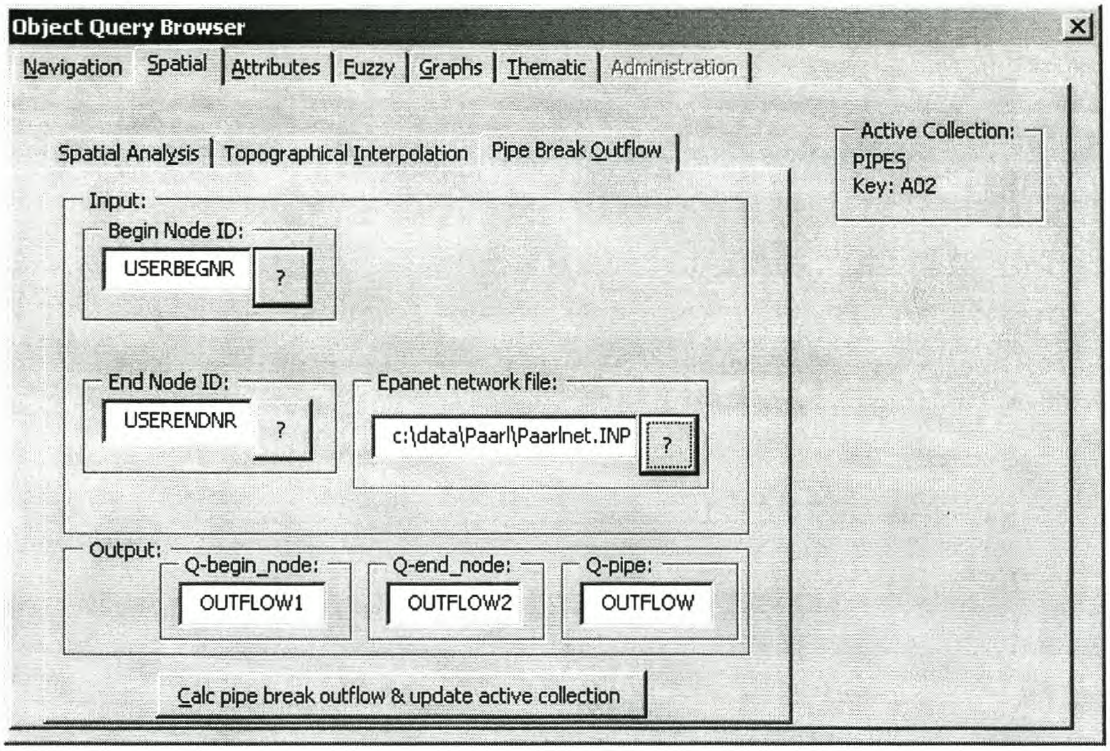
FIGURE 5.25 Topographical Interpolation sub-GUI: WADPOL Interpolator

5.4.2.3 Pipe break outflow

The water distribution system analysis program, EPANET version 2 from the U.S. Environmental Protection Agency, is used as the hydraulic engine for the pipe break outflow analysis. The EPANET program can be used either as a stand-alone executable program or its hydraulic functions can be called from other MS-Windows applications via the accompanying EPANET Programmer's Toolkit, a dynamic link library (DLL). The DLL functions declaration of the dynamic link library is contained in a Visual Basic module and can thus be added to the Object Query Browser VBA project. The EPANET hydraulic functions can then be called directly from the class in Object Query Browser responsible for the pipe break outflow analysis. An integrated and seamless environment for pipe break outflow analysis can thus be established within the SDSS. A description of the Pipe Break Outflow sub-GUI of the Spatial Analysis GUI (see Figure 5.26) will now follow:

- The attributes in the active query collection containing the begin and end node identification codes of the pipes must be selected in the *Input* box. The EPANET input file containing the pipe network information necessary for the hydraulic pipe break outflow analysis must also be specified in the *Input* box.

- The names of the attributes that will contain the results of the pipe break outflow analysis must be entered in the respective slots of the *Output* box. The pipe break outflow will be calculated by simulating breaks at the begin and end node of a pipe. The average outflow will then be assigned to the pipe (see Section 6.3.1). The Q_{begin_node} and Q_{end_node} attributes will contain the calculated outflow at the begin and end node of a pipe respectively. The Q_{pipe} attribute will contain the average outflow for the pipe.



Active Collection:
PIPES
Key: A02

Input:
Begin Node ID: USERBEGNR
End Node ID: USERENDNR
Epanet network file: c:\data\Paarl\Paarlnet.INP

Output:
Q-begin_node: OUTFLOW1
Q-end_node: OUTFLOW2
Q-pipe: OUTFLOW

FIGURE 5.26 Spatial GUI: Pipe Break Outflow sub-GUI

- The *Calc pipe break outflow & update active collectio*n button can be clicked to call the *calcOutflow* operation of the QueryTree class (see Figure 5.4), which will calculate the outflow from simulated pipe breaks for each pipe in the active query collection and will update the collection accordingly. The *calcOutflow* operation is a specialised operation for pipe break impact assessment and will be discussed in detail in Section 6.3.1.4.

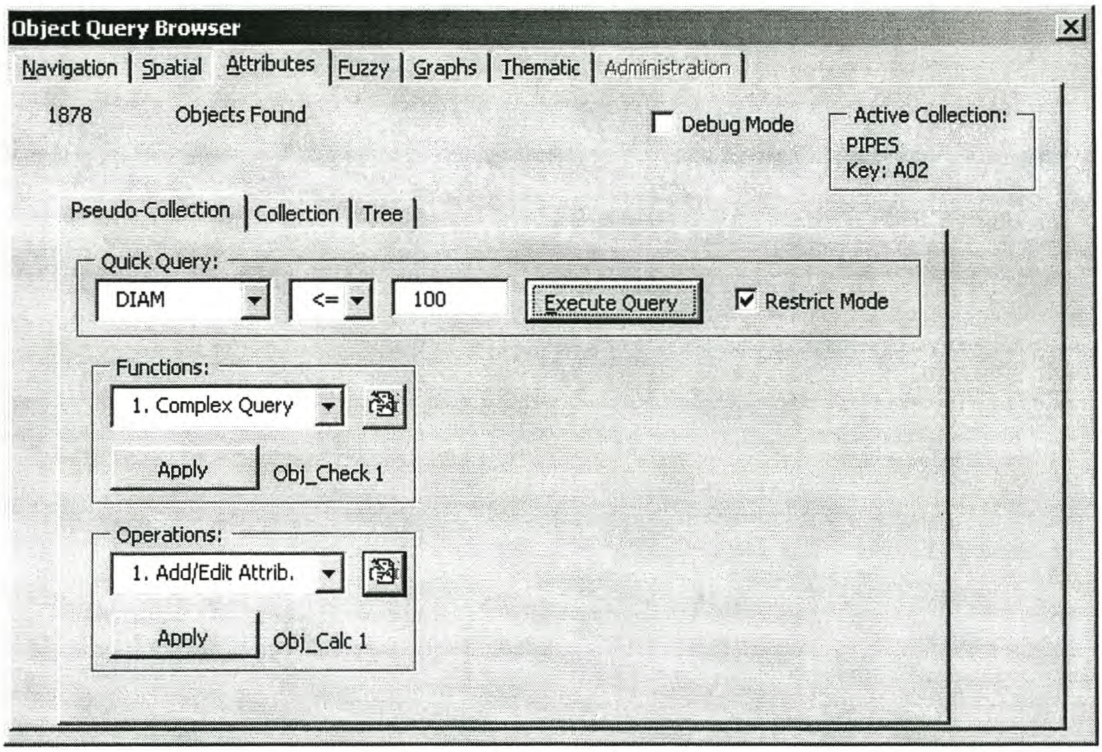
5.4.3 Attributes Graphical User Interface (GUI)

The Attributes GUI (see Figure 5.27) of the Object Query Browser provides access to the attribute query functions and manipulation operations for both simple and complex data analysis. The available query functions and manipulation operations in the Object Query Browser work on three different data access levels, viz. pseudo-collection level, collection level and tree level. The Attributes GUI consists of three sub-GUIs - one for each attribute data access level - and will be described in the sections that follow.

5.4.3.1 Pseudo-collection level attribute data access

The scope of the query functions and manipulation operations that can be called from the Pseudo-Collection sub-GUI of the Attributes GUI (see Figure 5.27) covers the whole active query collection and its associated collection, but access to the individual data items is limited. The system will automatically iterate through the active query collection and apply the specified function or specified operation to each object in the collection. Individual objects in the collection are inaccessible so that different functions or operations to be applied to these individual objects cannot be specified. The pseudo-collection level queries and manipulations are ideal for simpler checks or tasks that must be carried out on all objects in the active query collection. Although the pseudo-collection level queries have only limited access to the attributes, their main advantage over the other higher access level queries is that the iteration through the collection is managed by the system. The SDSS Analyst therefore does not have to specify intricate looping statements in the query. A description of the Pseudo-Collection sub-GUI (see Figure 5.27) will now follow:

- The *Execute Query* button in the *Quick Query* box can be clicked to execute simple queries on a single attribute and the following operators are supported: =, <, >, <=, >= and <>. When the *Restrict Mode* check box is checked (the default mode), then the active query collection will be restricted, i.e. reduced so that it contains only the objects that fall within the query. If the *Restrict Mode* check box is unchecked, then the objects found within the query will only be listed temporarily in the attribute table of the Navigation GUI and the active query collection will still contain all its objects.



Object Query Browser
Navigation | Spatial | Attributes | Fuzzy | Graphs | Thematic | Administration
Pseudo-Collection | Collection | Tree
Execute Query
Apply
Apply

1878 Objects Found

□ Debug Mode

Active Collection:
PIPES
Key: A02

Quick Query: DIAM <= 100

☑ Restrict Mode

Functions: 1. Complex Query
Obj_Check 1

Operations: 1. Add/Edit Attrib.
Obj_Calc 1

FIGURE 5.27 Object Query Browser: Attributes GUI & Pseudo-Collection sub-GUI

- The pseudo-collection access level further supports ten available functions *obj_Check1..10* of the TreeOperations class (see Figure 5.6), that can be selected from the *Functions* box to be used in more complex queries. The return type of these functions is of type Boolean – when true then the object will remain in the active query collection, when false then it will be removed from the collection. The small button in the *Functions* box can be clicked to edit the function identification name. The name change will only be reflected in the list that is shown in the *Functions* box, i.e. the internal signature of the function will not be changed. The *Apply* button in the *Functions* box can be clicked to apply the query function to each object in the active query collection. The resulting object interaction which then occurs closely resembles the object interaction for operations, which will be described under the next bullet. Function selection is not supported for the higher data access level queries, since equivalent if-statements and subsequent object removal statements will then be more appropriate.

- There are ten available operations *obj_Calc1..10* of the TreeOperations class (see Figure 5.6) that can be selected from the *Operations* box to manipulate the objects. The small button in the *Operations* box can be clicked to edit the operation identification name. The name change will only be reflected in the list that is shown in the *Operations* box, i.e. the internal operation signature will not be changed. The *Apply* button will apply the operation to each object in the active query collection. The sequence diagram (see Figure 5.28) shows the object interaction that occurs when the SDSS Analyst selects one of the pseudo-collection level operations *obj_Calc1..10* and then clicks the *Apply* button. The object interaction is basically between the *Apply* button of the user interface and an instance of the QueryTree class, as well as between an instance of the QueryTree class and an instance of the TreeOperation class, where the manipulations will be carried out on the object data (see Section 5.2.2). The object interaction that occurs when applying operations in the two higher data access levels (see Sections 5.4.3.2 and 5.4.3.3) closely resembles the sequence shown in Figure 5.28.

- The *Debug Mode* checkbox can be checked, so that when the *Apply* button is clicked, the program will temporarily stop and display the VBA Editor with the program cursor located in the source code listing of the selected function or operation. The SDSS Operator or SDSS Analyst can then customise the program source code in an indicated editable region. The program execution can then be continued by clicking the *Continue* button in the VBA Editor. This customisable program code of the Object Query Browser at run-time, offers an extremely flexible and rich query environment, but care should be taken when assigning user access rights (see Section 5.4.7) so that only qualified SDSS Operators and SDSS Analysts will have access to the Attributes GUI.

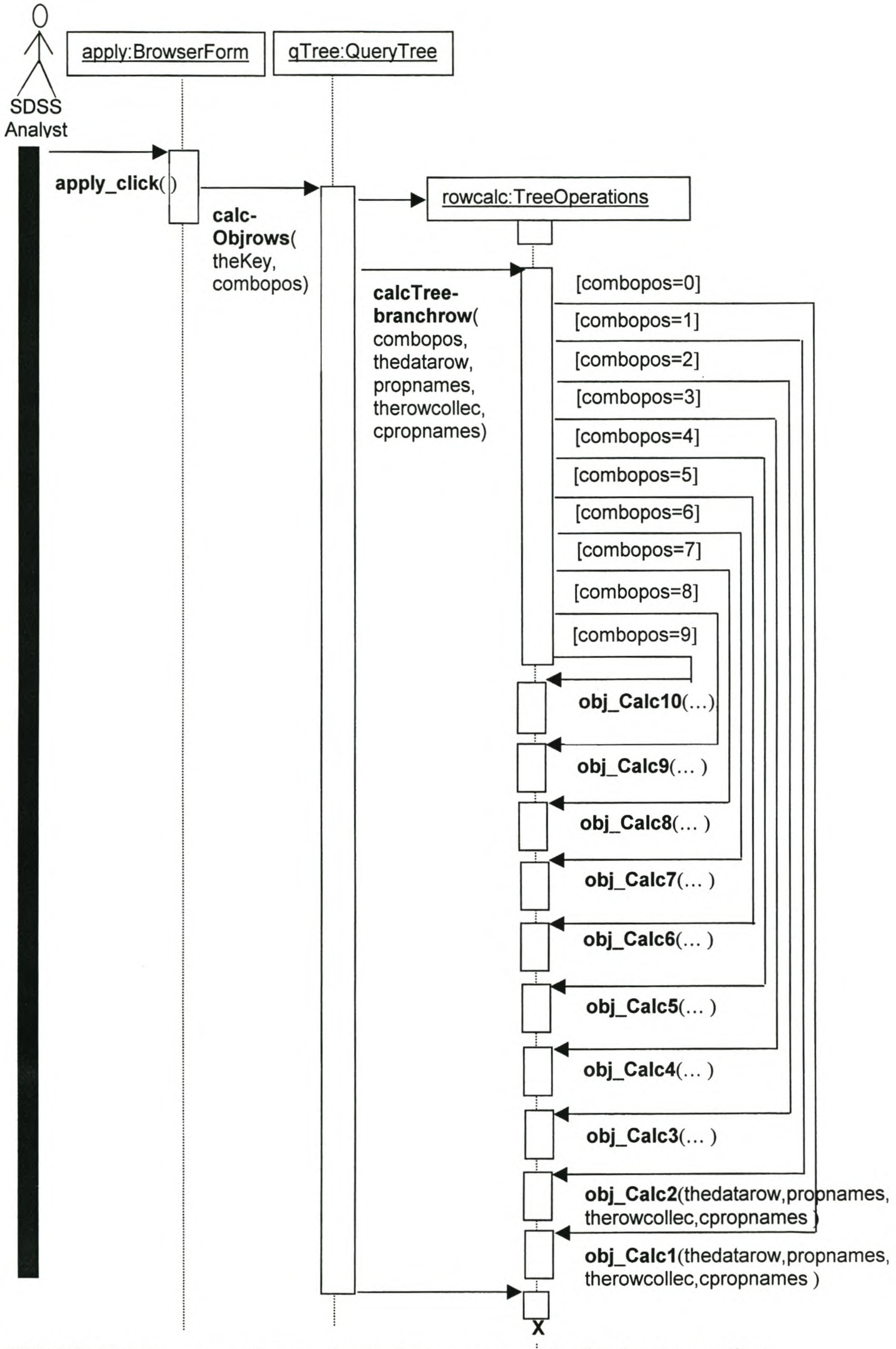


FIGURE 5.28 Sequence diagram for applying pseudo-collection level operations

Activity description: << **apply_click( )**>>

The click event of the *Apply* button activates when the SDSS Analyst clicks the *Apply* button (see Figure 5.27). The click event of the *Apply* button then calls the *calcObjrows* operation of QueryTree class and sends over the active query collection key and the combobox position of the selected operation.

Activity description:<< **calcObjrows**(theKey, combopos)>>

The *calcObjrows* operation of the QueryTree class, having received the active query collection key and combobox position, will access the corresponding query collection data from the private *queryRes* collection of the QueryTree class (see Figure 5.4). The *calcObjrows* operation also instantiates the *rowcalc* object of the TreeOperations class.

The *calcObjrows* operation then iterates through all objects in the active query collection. The combobox position indicating the selected operation and the attribute data for both the current object in the iteration, as well as its associated object, are passed on to the *rowcalc object* via the *calcTreebranchrow* operation call.

The *calcCollec* and *calcWholeTree* operations of the QueryTree class (see Figure 5.4) that have been defined for the two higher data access levels, viz. collection level and tree level, fulfil a similar function as the *calcObjrows* operation. The *calcCollec* and *calcWholeTree* operations, however, contain no looping statements, since the iteration must in this case, be specified as part of the query in the *collec_Calc* and *tree_Calc* operations of the TreeOperations class.

Activity description: << **calcTreebranchrow**(combopos, thedatarow,
propnames, therowcollec, cpropnames) >>

The *calcTreebranchrow operation of the rowcalc object* (an instance of the TreeOperations class), having received the combobox postion and attribute data, will branch (see Section 3.2.4) to the corresponding *obj_Calc* operation using the combobox position as reference (which is zero based).

Activity description:<< **obj_Calc1(**thedatarow, propnames,
therowcollec, cpropnames) >>

The particular *obj_Calc* operation of the *rowcalc* object that was called will, after having received the attribute data, manipulate the data according to the statements defined within the operation. If the debug mode option (which will be discussed in the final bullet of this section) was set on, then the operation execution will temporarily stop, so that the user can edit the statements. Operation execution can then be continued.

FIGURE 5.29 Sequence diagram descriptions for applying pseudo-collection level operations

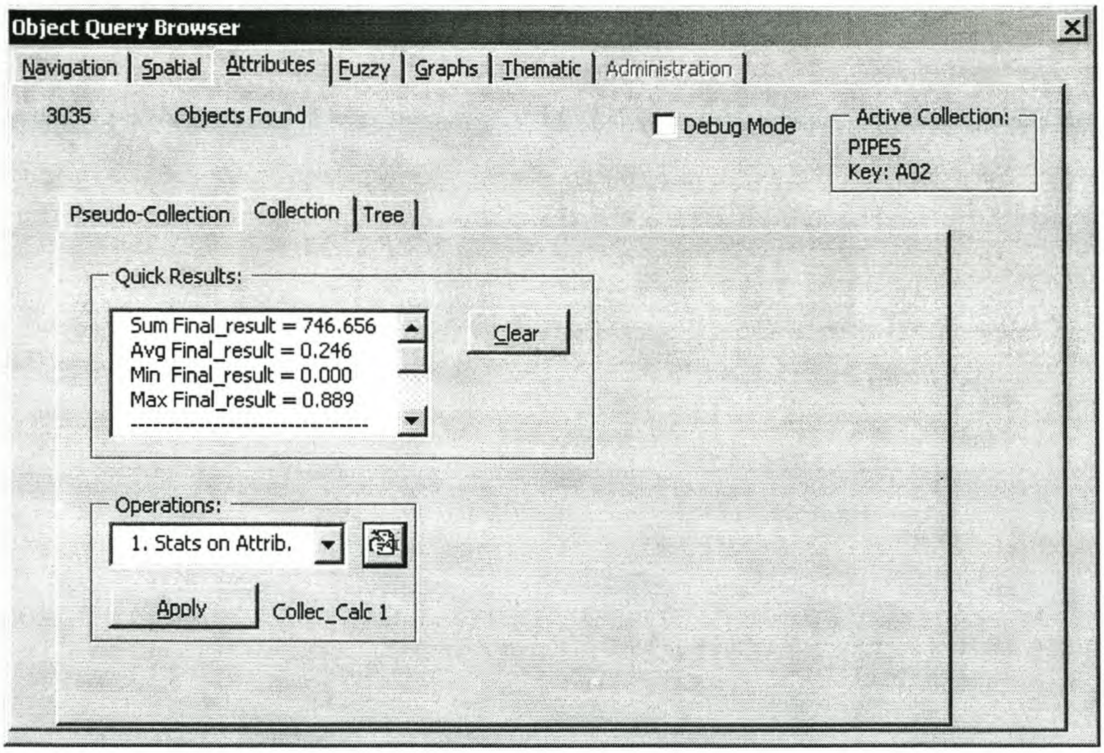
5.4.3.2 Collection level attribute data access

The scope of the operations that can be called from the Collection sub-GUI of the Attributes GUI (see Figure 5.30) also covers the whole active query collection and its associated collection, just as the pseudo-collection level operations, but the individual objects in the active query collection are now accessible. The user can also edit the iteration loops in the operation so that complex tasks can be carried out on all objects in the active query collection or on certain objects only. A description of the Collection sub-GUI (see Figure 5.30) will now follow:

- There are ten available operations *collec_Calc1..10* of the TreeOperations class (see Figure 5.6) that can be selected from the *Operations* box to manipulate the objects in the active query collection. The small button in the *Operations* box can be clicked to edit the operation identification name. The *Apply* button will apply the operation to the active query collection and similar object interaction will then occur as discussed in Section 5.4.3.1

- The *Debug mode* can be accessed in exactly the same way as discussed in Section 5.4.3.1

- The *collec_Calc1..10* operations can display calculated result figures in the *Quick Results* list box. The user can access the *collec_Calc* operations via the *Debug mode* option (see Section 5.4.3.1) and then specify what results are to be displayed in the *Quick Results* list box. The *Clear* button can be clicked to clear all items currently displayed in the *Quick Results* list box.



3035 Objects Found

☐ Debug Mode

Active Collection:
PIPES
Key: A02

```
Sum Final_result = 746.656
Avg Final_result = 0.246
Min  Final_result = 0.000
Max Final_result = 0.889
-------------------------------
```

1. Stats on Attrib.

Collec_Calc 1

FIGURE 5.30 Attributes GUI: Collection sub-GUI

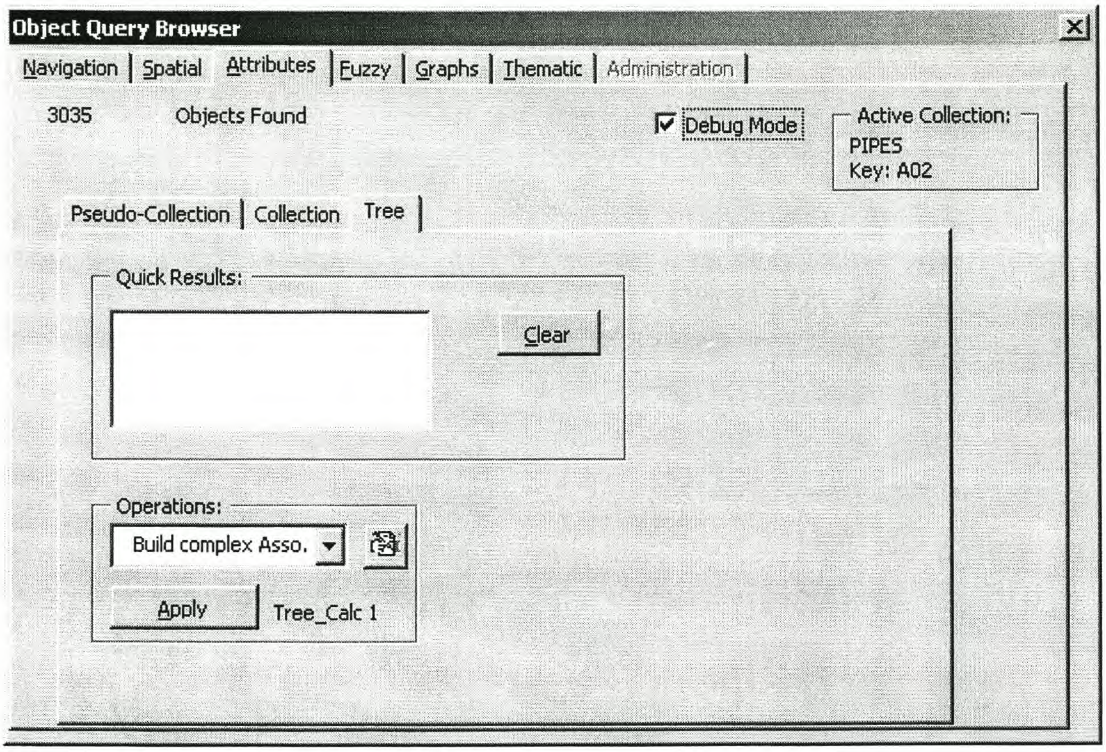
5.4.3.3 Tree level attribute data access

The scope of the operations that can be called from the Tree sub-GUI of the Attributes GUI (see Figure 5.31) covers all the query collections that are loaded into the Object Query Browser. Individual objects in all the query collections are accessible. The user can also edit the iteration loops in the operation so that complex tasks can be carried out on all objects in the Object Query Browser or on certain objects only. A description of the Tree sub-GUI (see Figure 5.31) will now follow:

- There are ten available operations *tree_Calc1..10* of the TreeOperations class (see Figure 5.6) that can be selected from the *Operations* box to manipulate the objects in the query collections. The small button in the *Operations* box can be clicked to edit the operation identification name. The *Apply* button will activate similar object interaction as for the operations that function on the two lower levels.

- The *Debug mode* can be accessed in the same way as discussed in Section 5.4.3.1.

- Results can be displayed in the *Quick Results* list box in the same way as discussed in Section 5.4.3.2.



3035 Objects Found

☑ Debug Mode

Active Collection:
PIPES
Key: A02

Build complex Asso. ▾

Tree_Calc 1

FIGURE 5.31 Attributes GUI: Tree sub-GUI

5.4.4 Fuzzy Graphical User Interface (GUI)

The Object Query Browser functionality is extended to support fuzzy-logic-based analysis by linking it with the fuzzy controller A-B Flex version 2.50a from Allen-Bradley Co.

A-B Flex provides functions that enable accessing of its services from other MS-Windows applications using the dynamic data exchange (DDE) protocol. A-B Flex can be used as a DDE server application that is accessed from multiple client applications like Microsoft Excel, Visual Basic, or MATLAB.

The academic version 2.50a of A-B Flex, however, is restricted by allowing the DDE server to be accessible only from Microsoft Excel 97. The Object Query Browser will therefore have to be linked with Microsoft Excel 97 in order to be a client of the A-B Flex DDE server. This interim link with Microsoft Excel can be avoided by installing the professional version of A-B Flex.

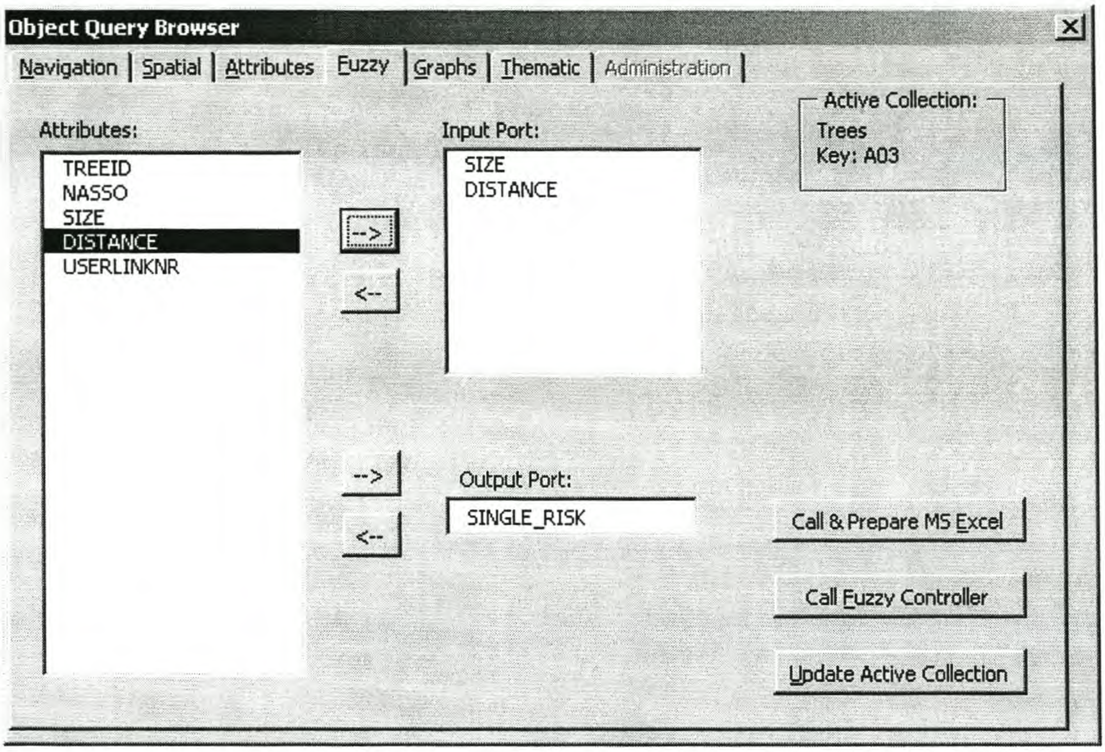
A description of the Fuzzy GUI (see Figure 5.32), which establishes the interim link with Microsoft Excel that is needed for the communication with the A-B Flex fuzzy controller, will now follow:

- The attribute names of the objects in the active query collection are listed in the *Attributes* box on the left side of the GUI.

- The attributes to be used as input and output variables in the fuzzy inference should be specified in the *Input Port* and *Output Port* boxes.

- The arrow buttons can be used to copy selected attributes over from the *Attributes* box to the *Input Port* and *Output P*ort boxes (the inverse, viz. to deselect the attributes, is also possible).

- The *Call & Prepare MS Excel* button can be clicked to access and prepare Microsoft Excel for the communication with the fuzzy controller. The attribute values of the objects in the active query collection to be used as inputs in the fuzzy controller will then be transferred to the *Fuzzytab.xls* spreadsheet (see Figure 5.33) via the *updateFuzzyTable* operation of the QueryTree class (see Figure 5.4). The *Fuzzytab.xls* spreadsheet is a template that contains the communication interface with the A-B Flex fuzzy controller and will automatically be loaded and filled with the attribute data values of the current query collection, via ActiveX client/server interaction. Applications that support ActiveX technology, such as Microsoft Excel, provide objects that can be manipulated programmatically from within external programming applications. The properties (such as the cells),



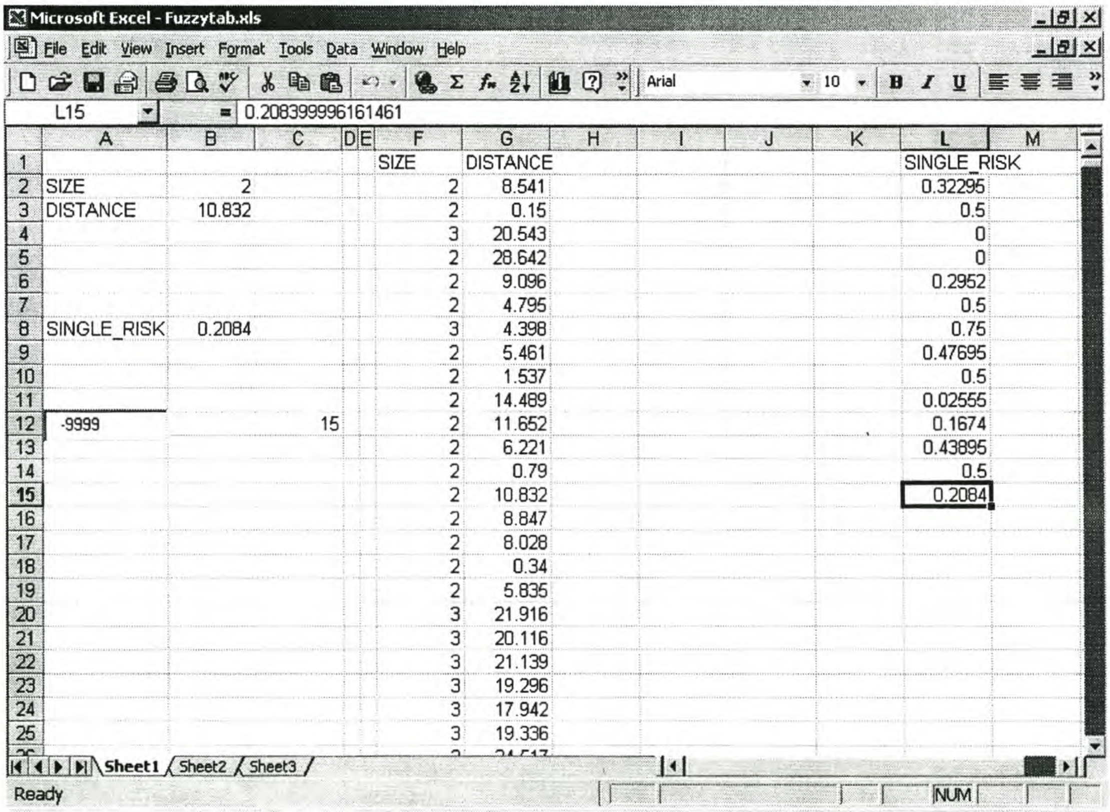
Attributes:
TREEID
NASSO
SIZE
DISTANCE
USERLINKNR

Input Port:
SIZE
DISTANCE

Output Port:
SINGLE_RISK

Active Collection:
Trees
Key: A03

FIGURE 5.32 Object Query Browser: Fuzzy GUI


Microsoft Excel - Fuzzytab.xls
File Edit View Insert Format Tools Data Window Help
Arial
10
Sheet1
Sheet2
Sheet3
Ready
NUM

L15 = 0.208399996161461

	A	B	C	D	E	F	G	H	I	J	K	L	M
1						SIZE	DISTANCE					SINGLE_RISK	
2	SIZE	2				2	8.541					0.32295	
3	DISTANCE	10.832				2	0.15					0.5	
4						3	20.543					0	
5						2	28.642					0	
6						2	9.096					0.2952	
7						2	4.795					0.5	
8	SINGLE_RISK	0.2084				3	4.398					0.75	
9						2	5.461					0.47695	
10						2	1.537					0.5	
11						2	14.489					0.02555	
12	-9999		15			2	11.652					0.1674	
13						2	6.221					0.43895	
14						2	0.79					0.5	
15						2	10.832					0.2084	
16						2	8.847						
17						2	8.028						
18						2	0.34						
19						2	5.835						
20						3	21.916						
21						3	20.116						
22						3	21.139						
23						3	19.296						
24						3	17.942						
25						3	19.336						

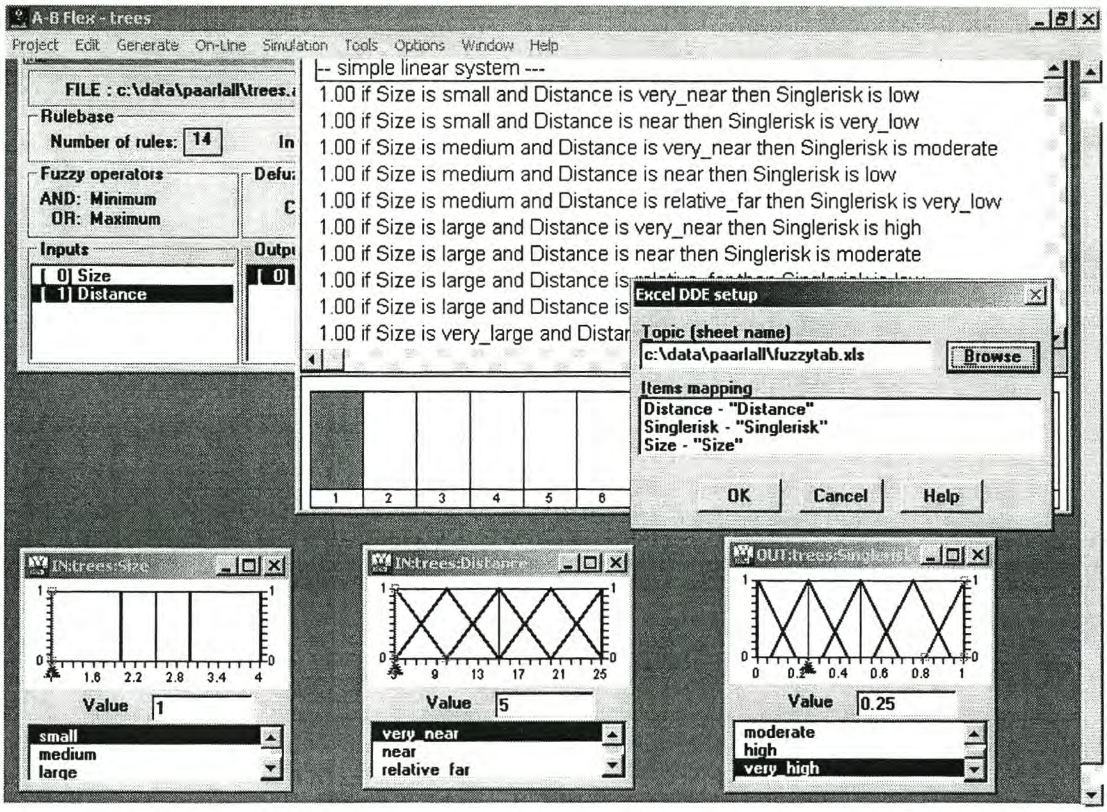
FIGURE 5.33 Fuzzy GUI: Microsoft Excel - *Fuzzytab.xls*

methods and events of the Microsoft Excel spreadsheet stored in *Fuzzytab.xls,* can therefore directly be accessed by the ActiveX statements in the *updateFuzzyTable* operation.

- The A-B Flex fuzzy controller (see Figure 5.34) can be accessed by clicking the *Call Fuzzy Controller* button (see Figure 5.32). The A-B Flex fuzzy controller will then open, as well as the last (frequently used) project, since the A-B Flex fuzzy controller has an option to automatically load in the last project at start-up. Changes can then be made to the project such as editing the fuzzy membership functions and the rule base (see Section 2.2.2). A surface plot (see Section 2.2.3) can also be drawn to obtain a graphical representation of the fuzzy rule base. The DDE set-up should be checked (see Figure 5.34) so that the input and output ports are mapped to the correct cells in the spreadsheet. The time-interval of data sampling during data exchange can also be adjusted in the DDE set-up box.

- The fuzzy inference algorithm can then be started from the A-B Flex *Simulation* menu. Dynamic data exchange between the A-B Flex fuzzy controller and the Microsoft Excel spreadsheet will then occur in the following way: A-B Flex receives the input values from the spreadsheet, performs the fuzzy inference and sends the resulting output value back to the spreadsheet. The input values of the next line in the spreadsheet are then sent to A-B Flex for fuzzy inference and the process continues in time-intervals as specified in the DDE set-up. A snapshot of this dynamic data exchange can be seen in Figure 5.33 where the fuzzy inference result (viz. output variable *Single_risk* = 0.2084) for the 15th row in the *Fuzzytab.xls* spreadsheet has just been calculated by A-B Flex.

- The fuzzy inference results that were calculated by the A-B Flex fuzzy controller and transferred to the Microsoft Excel spreadsheet via DDE, can finally be transferred to the active query results collection by clicking the *Update Active Collection* button (see Figure 5.32). The click event of this button calls the *updateFuzzyResults* operation of the QueryTree class (see Figure 5.4), containing the necessary ActiveX statements to access the *Fuzzytab.xls* result cells.



FILE : c:\data\paarlall\trees.

Rulebase
Number of rules: 14

Fuzzy operators
AND: Minimum
OR: Maximum

Inputs
[0] Size
[1] Distance

```
-- simple linear system ---
1.00 if Size is small and Distance is very_near then Singlerisk is low
1.00 if Size is small and Distance is near then Singlerisk is very_low
1.00 if Size is medium and Distance is very_near then Singlerisk is moderate
1.00 if Size is medium and Distance is near then Singlerisk is low
1.00 if Size is medium and Distance is relative_far then Singlerisk is very_low
1.00 if Size is large and Distance is very_near then Singlerisk is high
1.00 if Size is large and Distance is near then Singlerisk is moderate
1.00 if Size is large and Distance is
1.00 if Size is large and Distance is
1.00 if Size is very_large and Distar
```

Topic (sheet name): c:\data\paarlall\fuzzytab.xls

Items mapping
Distance - "Distance"
Singlerisk - "Singlerisk"
Size - "Size"

Value 1
small
medium
large

Value 5
very_near
near
relative_far

Value 0.25
moderate
high
very_high

FIGURE 5.34 Fuzzy GUI: A-B Flex fuzzy controller

5.4.5 Graphs Graphical User Interface (GUI)

For the graphical representation of the query results, the Object Query Browser of the SDSS is linked with Microsoft Excel 97 via the Graphs GUI shown in Figure 5.35. The internal VBA graphing functions could have been used instead to be independent from Microsoft Excel, but the sophisticated graphing functionality available in Microsoft Excel would be very difficult to match. A description of the Graphs GUI (see Figure 5.35) will now follow:

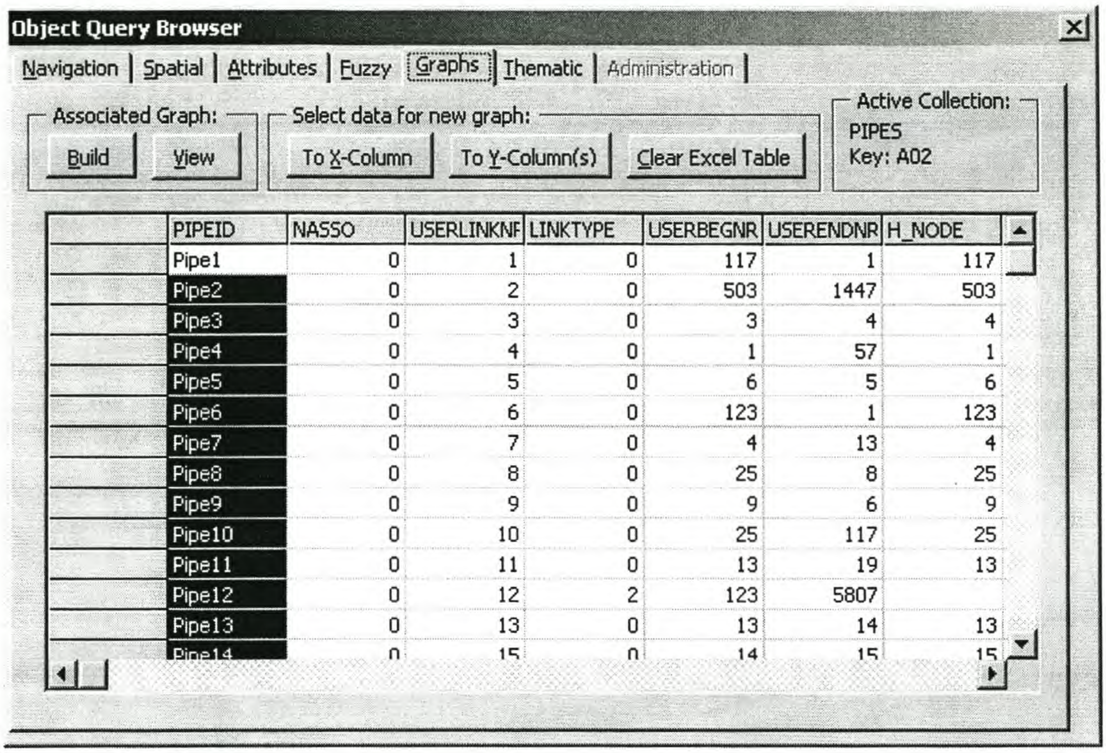
- The column containing the independent x-axis data to be graphed should be selected as indicated in Figure 5.35. The *To X-Column* button can then be clicked to transfer the data column to the Microsoft Excel spreadsheet *GraphView.xls* (see Figure 5.36), which is a template for preparing the graphs. The data transfer is based on the same ActiveX client/server interaction discussed in Section 5.4.4.

- The *To Y-Column(s)* button in the Graphs GUI can then be clicked to transfer the data column containing the dependent y-axis data to *GraphView.xls* (also via ActiveX client/server interaction). Y-axis data columns can be copied repeatedly one after the other to *GraphView.xls* if a graph containing more than one data series is to be compiled.

- When the data transfer is completed the graphs will be automatically generated in *GraphView.xls* (see Figure 5.36) and can then be customised. The chart type (bar graphs, line graphs, etc.), source data (data range, series, etc.), chart options (titles, axis, gridlines, legends, etc.) and chart location (on a separate chart sheet or as an object in an existing sheet, etc.) can be edited using the standard Microsoft Excel chart wizard.

- The statistics on the y-axis values, such as sum, average, min and max are automatically calculated and presented in *GraphView.xls* (see Figure 5.36).

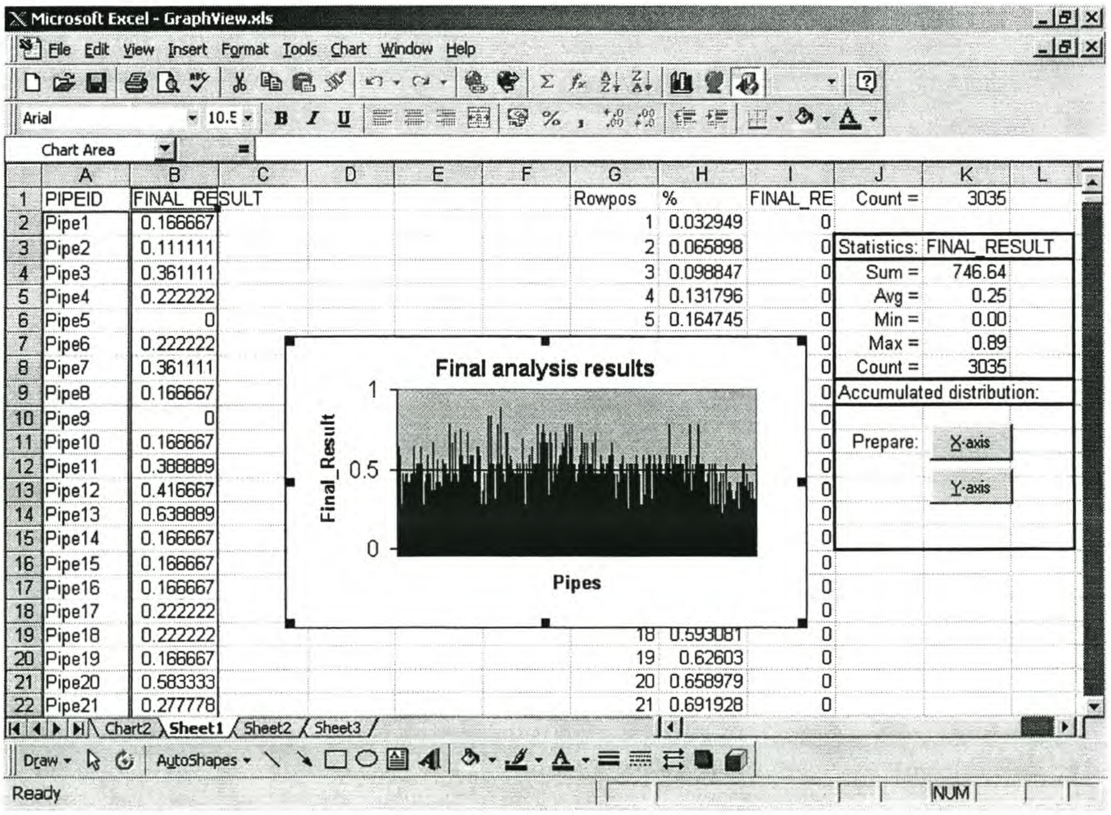
- The *X-axis* button in *GraphView.xls* (see Figure 5.36) can be clicked to prepare the x-axis data for a cumulative distribution plot. The x-axis data values are the percentages of the total number of data. The *Y-axis* button in the *GraphView.xls* (see Figure 5.36) can be clicked to obtain a data column containing the sorted y-axis data values for the cumulative distribution plot. Chapter 7 contains many examples of cumulative distribution plots that have been produced with *GraphView.xls*.



Active Collection:
PIPES
Key: A02

PIPEID	NASSO	USERLINKNF	LINKTYPE	USERBEGNR	USERENDNR	H_NODE
Pipe1	0	1	0	117	1	117
Pipe2	0	2	0	503	1447	503
Pipe3	0	3	0	3	4	4
Pipe4	0	4	0	1	57	1
Pipe5	0	5	0	6	5	6
Pipe6	0	6	0	123	1	123
Pipe7	0	7	0	4	13	4
Pipe8	0	8	0	25	8	25
Pipe9	0	9	0	9	6	9
Pipe10	0	10	0	25	117	25
Pipe11	0	11	0	13	19	13
Pipe12	0	12	2	123	5807	
Pipe13	0	13	0	13	14	13
Pipe14	0	15	0	14	15	15

FIGURE 5.35 Object Query Browser: Graphs GUI



	A	B	C	D	E	F	G	H	I	J	K	L
1	PIPEID	FINAL_RESULT					Rowpos	%	FINAL_RE	Count =	3035	
2	Pipe1	0.166667					1	0.032949	0			
3	Pipe2	0.111111					2	0.065898	0	Statistics:	FINAL_RESULT	
4	Pipe3	0.361111					3	0.098847	0	Sum =	746.64	
5	Pipe4	0.222222					4	0.131796	0	Avg =	0.25	
6	Pipe5	0					5	0.164745	0	Min =	0.00	
7	Pipe6	0.222222							0	Max =	0.89	
8	Pipe7	0.361111							0	Count =	3035	
9	Pipe8	0.166667							0	Accumulated distribution:		
10	Pipe9	0							0			
11	Pipe10	0.166667							0	Prepare:	X-axis	
12	Pipe11	0.388889							0			
13	Pipe12	0.416667							0		Y-axis	
14	Pipe13	0.638889							0			
15	Pipe14	0.166667							0			
16	Pipe15	0.166667							0			
17	Pipe16	0.166667							0			
18	Pipe17	0.222222							0			
19	Pipe18	0.222222					18	0.593081	0			
20	Pipe19	0.166667					19	0.62603	0			
21	Pipe20	0.583333					20	0.658979	0			
22	Pipe21	0.277778					21	0.691928	0			

FIGURE 5.36 Graphs GUI: Microsoft Excel - *GraphView.xls*

- The *GraphView.xls* spreadsheet can be saved separately under a different filename that can then be associated with the current query collection by clicking the *Build* button in the Graphs GUI. The name of the associated Microsoft Excel spreadsheet, containing the associated graph(s) and statistics of the current query collection, is then stored in the *graphName* attribute of the Treebranch class (see Figure 5.4).

- The *View* button in the Graphs GUI can be clicked to view the associated precompiled graph(s) and statistics of the current query collection. Microsoft Excel will automatically then start up and open the associated spreadsheet. This has been accomplished by ActiveX statements that access the associated filename stored in the *graphName* attribute of the current query collection. The *View* button facility has been designed mainly for the Public Works Administrator, who would want to have instant access to the associated graphs and statistics of a query results collection.

- The *Clear Excel Table* button in the Graphs GUI can be clicked to initialise the *GraphView.xls* template, i.e. clear all previously specified data so that a new graph can be drawn.

5.4.6 Thematic Graphical User Interface (GUI)

Thematic mapping functionality to create thematic maps of the query results can be accessed from the Thematic GUI (see Figure 5.37) of the Object Query Browser. The Thematic GUI links the SDSS with ArcView GIS, where the thematic mapping can be performed. A description of the Thematic GUI (see Figure 5.37) will now follow:

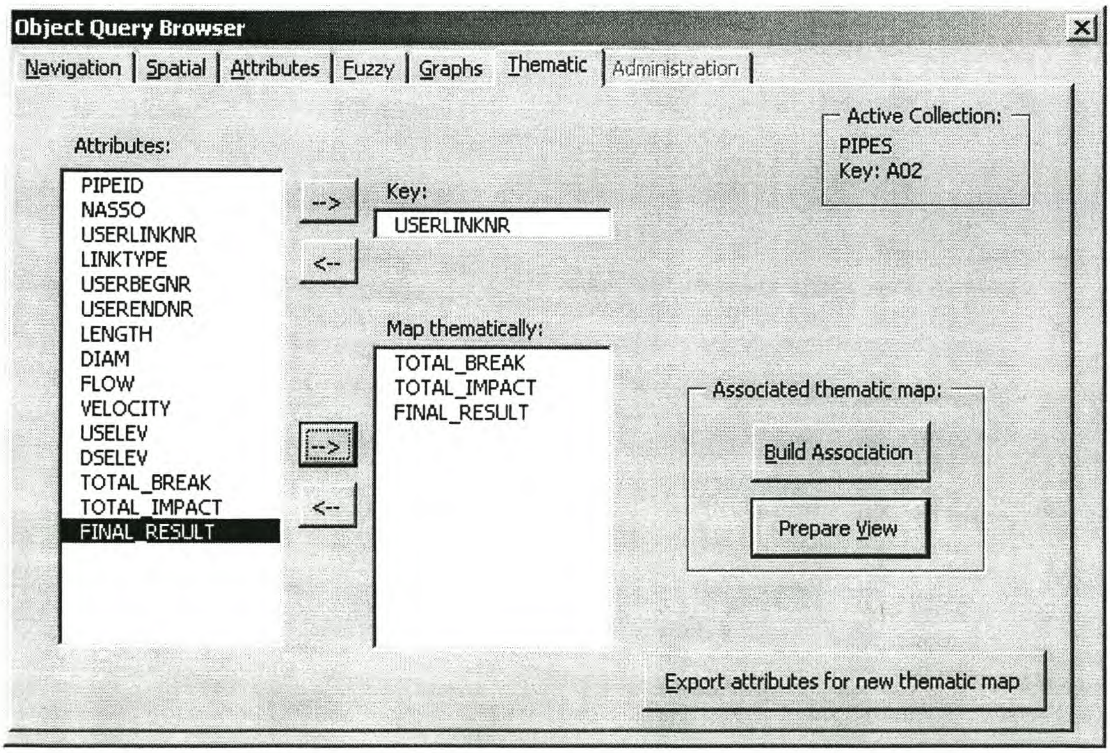
- The attribute names of the objects in the active query collection are listed in the *Attributes* box on the left side of the Thematic GUI.

- The attribute to be used as key to link the attributes with the spatial entities in ArcView (see Section 5.4.2.1) should be specified in the *Key* box.

- The attributes to be mapped thematically in ArcView should be specified in the *Map thematically* box.

- The arrow buttons can be used to select and copy the attributes listed in the *Attributes* box over to the *Key* box and the *Map thematically* box (the inverse, viz. to deselect the attributes, is also possible).

- The attribute data (including the key data) to be mapped thematically are then transferred to ArcView by clicking the *Export attributes for new thematic map* button.



Attributes:
PIPEID
NASSO
USERLINKNR
LINKTYPE
USERBEGNR
USERENDNR
LENGTH
DIAM
FLOW
VELOCITY
USELEV
DSELEV
TOTAL_BREAK
TOTAL_IMPACT
FINAL_RESULT

Key: USERLINKNR

Map thematically:
TOTAL_BREAK
TOTAL_IMPACT
FINAL_RESULT

Active Collection:
PIPES
Key: A02

FIGURE 5.37 Object Query Browser: Thematic GUI

- The spatial theme should be selected in ArcView on which the thematic mapping will be based. The first of the four customised ArcView buttons that are located in the top right corner above the scale box, with the icon (see Figure 5.38), can be clicked to join the basic attributes (consisting of only the spatial entity identifier, see Section 5.4.2.1) of the spatial entities with the attributes from the Object Query Browser. Copies of the selected spatial theme can be made and unique theme names can be given for these copies by applying the customised ArcView button with the icon (see Figure 5.38). Thematic mapping on any one of the joined attributes, in the original or copied themes, can then be accomplished by applying the standard ArcView thematic mapping tools.

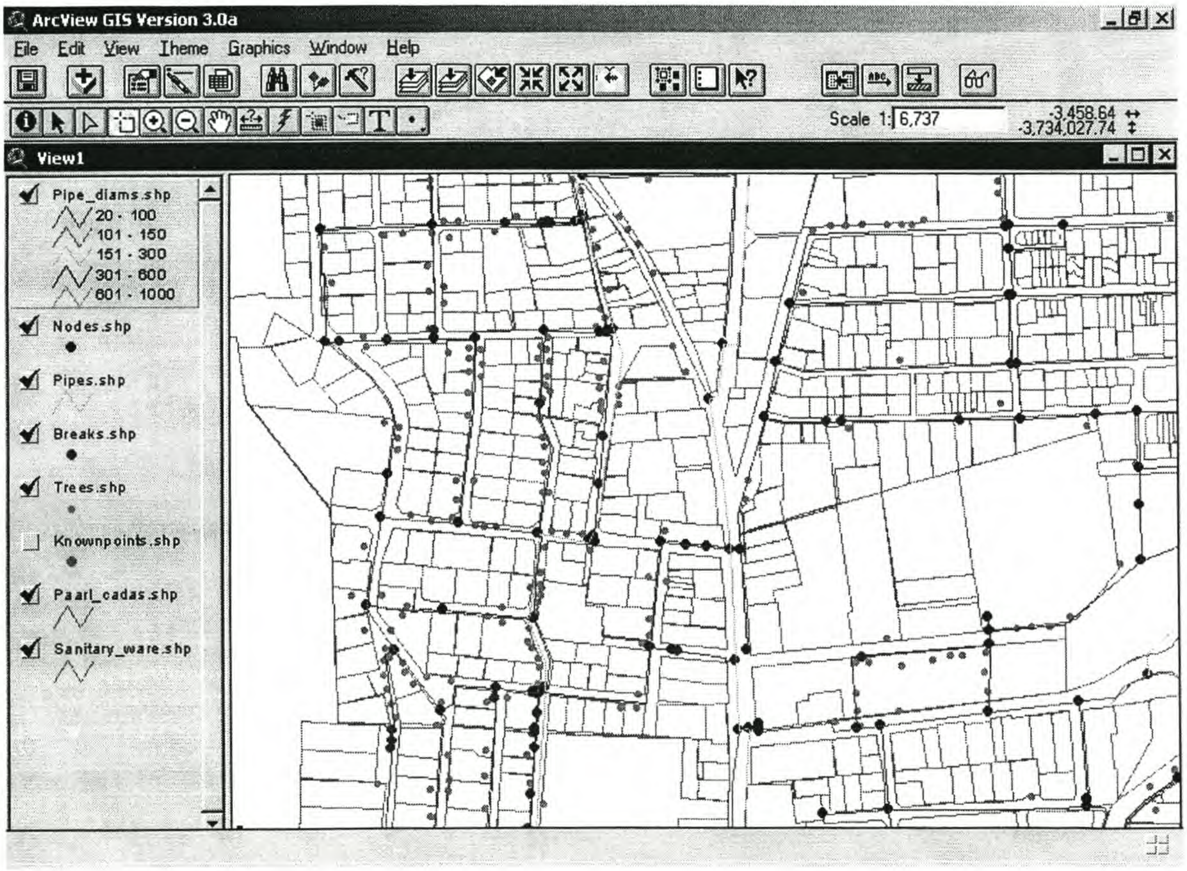


FIGURE 5.38 Thematic mapping of query results in ArcView

- The newly created thematic map theme can be associated with the current query collection by clicking the *Build Association* button in the Thematic GUI. The name of the associated thematic map theme of current query collection is then stored in the attribute *mapName* of the Treebranch class (see Figure 5.4).

- The functionality offered by the *Prepare View* button in the Thematic GUI has been designed mainly for the Public Works Administrator to facilitate access to associated thematic maps. The Public Works Administrator can click the *Prepare View* button in the Thematic GUI to prepare viewing the associated precompiled thematic map of the current query collection. The name of the associated thematic map theme that is stored in the *mapName* attribute of the current query collection will then be transferred to ArcView. In ArcView the Public Works Administrator can click on the customised button with the icon (see Figure 5.38) to view the associated thematic map. The thematic map theme will automatically move to the top of the theme list, so that there are no other themes to obscure its visibility.

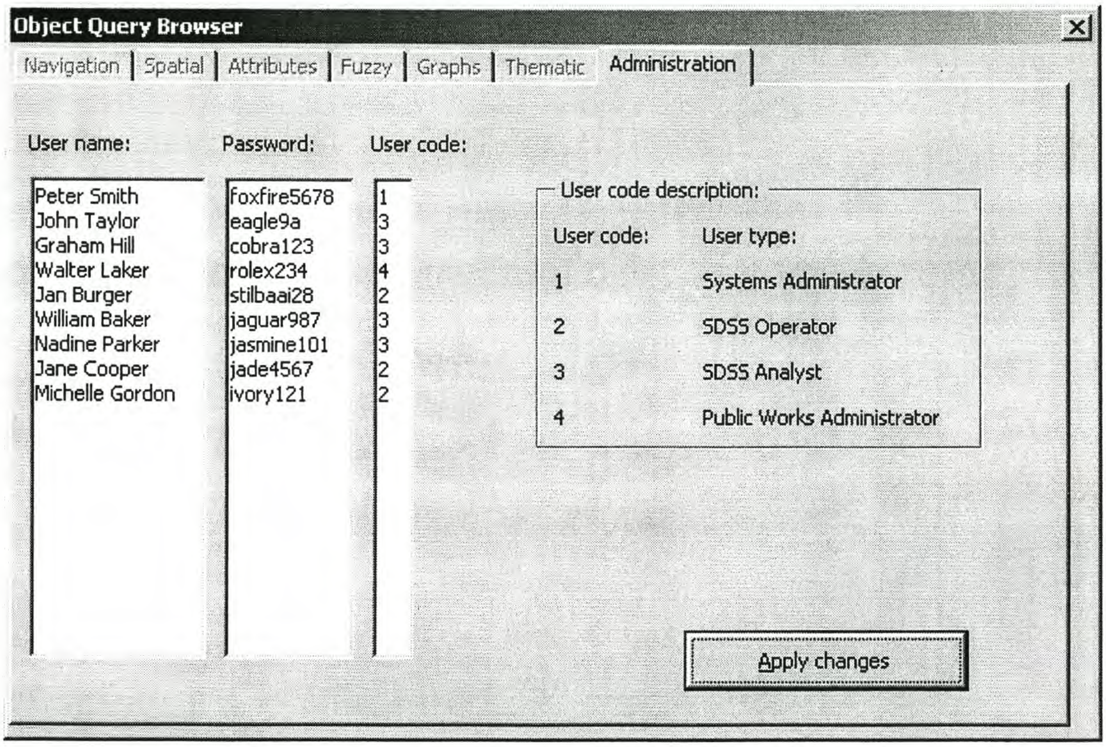
5.4.7 Administration Graphical User Interface (GUI)

The Administration GUI of the Object Query Browser (see Figure 5.39) is only accessible to systems administrators, i.e. to users that login to the system (see Figure 5.18) as Systems Administrator. A description of the Administration GUI (see Figure 5.39) will now follow:

- When the Administration GUI opens, the SDSS user information is read in from a hidden text file and appears in a table that lists the following information for each user: the user name, password and user code. The user code is used to specify the type of user and can be one of the following: 1 - Systems Administrator, 2 - SDSS Operator, 3 - SDSS Analyst, or 4 - Public Works Administrator.

- The Systems Administrator can then add new users, remove users, or change the particulars of a user by editing the contents of the *User name*, *Password* and *User code* boxes.

- The changes are made persistent by clicking the *Apply changes* button that will overwrite the hidden text file with the new user information.



User name:	Password:	User code:
Peter Smith	foxfire5678	1
John Taylor	eagle9a	3
Graham Hill	cobra123	3
Walter Laker	rolex234	4
Jan Burger	stilbaai28	2
William Baker	jaguar987	3
Nadine Parker	jasmine101	3
Jane Cooper	jade4567	2
Michelle Gordon	ivory121	2

User code description:

User code:	User type:
1	Systems Administrator
2	SDSS Operator
3	SDSS Analyst
4	Public Works Administrator

FIGURE 5.39 Object Query Browser: Administration GUI

The assigned user type will give a user access rights to the following GUIs of the Object Query Browser as indicated in Table 5.3.

TABLE 5.3 User access rights to the Object Query Browser GUIs

User type	**Accessible GUI(s)**
Systems Administrator	Administration
SDSS Operator	Navigation, Spatial, Attributes, Thematic
SDSS Analyst	Navigation, Spatial, Attributes, Fuzzy, Graphs, Thematic
Public Works Administrator	Navigation, Graphs, Thematic

Every time the Object Query Browser starts up, a login box will appear (see Figure 5.18) where the user name and corresponding password must be entered. The system will look up the access rights for the specified user in the hidden text file and will then adapt the GUIs accordingly, i.e. disable (dim out) the GUIs that are not accessible to the specific type of user. If the user name and password combination is invalid, then access to the Object Query Browser will be denied. The user name and password of the Systems Administrator have to be specified during installation of the Object Query Browser and will be the first entry in the hidden text file.

5.5 SUMMARY

In this chapter the general user requirements, the general system architecture, the general system functionality and the user interface of the SDSS have been outlined and designed. The SDSS is designed according to object-oriented modelling concepts and the system is composed of various subsystems. The Object Query Browser and a GIS (viz. ArcView) are the two basic subsystems. The Object Query Browser can be regarded as the control panel of the SDSS from where all the query, manipulation and display functionality can be accessed. The GIS subsystem provides the spatial query, spatial selection and thematic mapping functionality. The user interface of the SDSS that has been designed in this chapter, consists of seven GUIs and resides in the Object Query Browser of the SDSS. In the next chapter more detail will be added to the general design of the SDSS - subsystems for pipe break susceptibility analysis and impact assessment will then be designed.

CHAPTER 6: THE SUBSYSTEMS OF A SPATIAL DECISION SUPPORT SYSTEM FOR PIPE BREAK SUSCEPTIBILITY ANALYSIS AND IMPACT ASSESSMENT

In this chapter the specialised functions and operations needed for the detailed modelling of pipe break susceptibility analysis and impact assessment will be designed and integrated with the general query, manipulation and display functionality of the SDSS that was discussed and designed in Chapter 5.

6.1 BASIC USER REQUIREMENTS OF THE SUBSYSTEMS OF THE SPATIAL DECISION SUPPORT SYSTEM

As seen from a use case architectural modelling view (see Section 3.2), the SDSS for pipe break susceptibility analysis and impact assessment can be grouped into the subsystems as indicated by the UML Packages shown in Figure 6.1. This arrangement of subsystems is basically a further subdivision of the basic user requirements model that was shown in Figure 5.1 and more detail is added to the model. The subsystems on the lowest level shown in Figure 6.1 will model the detailed pipe break susceptibility analysis due to age, air pockets and trees; they will also model the detailed pipe break impact assessment with regard to water loss and property damage.

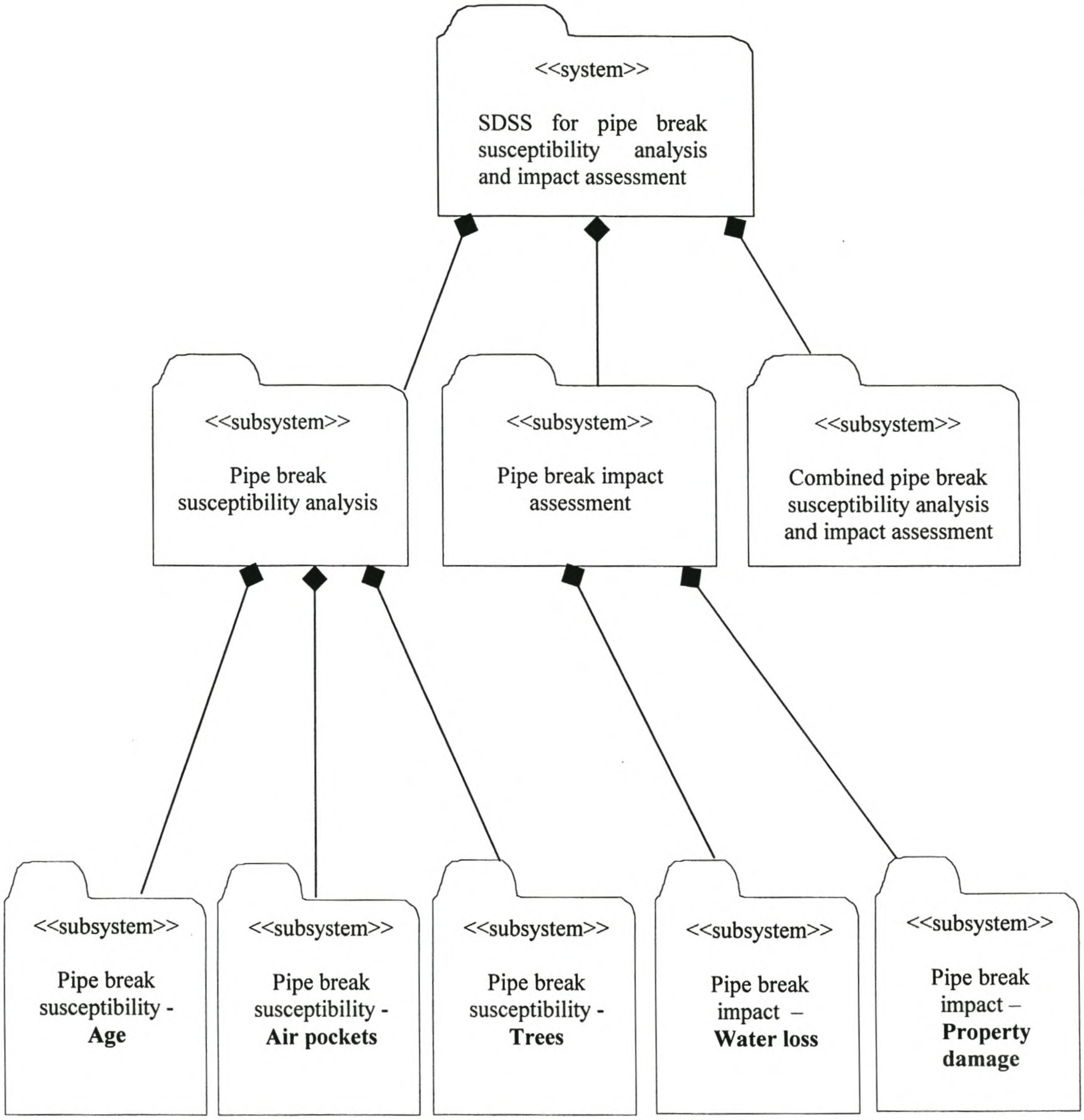

<<system>>
SDSS for pipe break susceptibility analysis and impact assessment
<<subsystem>>
Pipe break susceptibility analysis
<<subsystem>>
Pipe break impact assessment
<<subsystem>>
Combined pipe break susceptibility analysis and impact assessment
<<subsystem>>
Pipe break susceptibility - Age
<<subsystem>>
Pipe break susceptibility - Air pockets
<<subsystem>>
Pipe break susceptibility - Trees
<<subsystem>>
Pipe break impact – Water loss
<<subsystem>>
Pipe break impact – Property damage

FIGURE 6.1 Use case architectural modelling view of the subsystems of the SDSS

6.2 REQUIREMENTS ANALYSIS AND DESIGN OF SUBSYSTEMS FOR PIPE BREAK SUSCEPTIBILITY ANALYSIS

In this section the detailed requirements analysis and design of the subsystems of the SDSS will be given for analysing potential pipe breakage based on the pipe break theory outlined in Chapter 4. It would eventually be possible to model all the pipe break causes pointed out in Chapter 4. However, at this stage, with the digital data that was available on the study area of this dissertation (see Chapter 7), only three pipe break causes can be modelled accurately. The pipe break susceptibility analyses, taking into account the effects of age, air pockets and trees will now be discussed separately under the sections that follow.

6.2.1 Susceptibility to breakage due to age

Pipe age cannot always be determined precisely, because the old network installation plans are often not available any longer. Pipe replacement dates are often written in paper logbooks and must then be converted into digital format, which is a tedious process. The year of inauguration for residential areas, however, is a more readily available figure and can give a good indication of the age of the pipes in that area. In new residential areas the pipes will definitely still be new and, because of the short period of possible deterioration, they will mostly all have low pipe break susceptibility (see Section 4.1). In old residential areas many pipes may have been rehabilitated recently or replaced, but there will usually still be more old pipes in use than new ones. The classification of *old* pipes is, however, not as precise and certain as the *new* pipes classification. Pipes can be considered as still new if they are younger than 25 years. This should not be taken as a rule and is largely dependent on the type of environment the pipes have been in contact with (see Section 4.1). The pipe break pattern of the pipes in the study area of this dissertation, however, seems to follow the 25 year rule (see Section 7.3.1). The results from an age analysis should never be used on their own because of the above-mentioned inaccuracy and the many other pipe break factors that may exist. The results from an age analysis should therefore be used in conjunction with the results from the other two pipe break susceptibility analyses that will be discussed later in this chapter. The results can be combined (superimposed) by the analysis that will be described in Section 6.4.

6.2.1.1 Requirements analysis

The SDSS Operator (see Figure 6.2) should first perform thematic mapping (see Section 5.4.6) on the inauguration year attribute of the *Res_Areas* collection (see Section 7.1.1 for data accuracy levels) to obtain an overview of the spatial distribution of these figures.

The SDSS Operator can then apply the ArcView *Select by Theme* tool to automatically select all pipes that are within a selected residential area. The selection can then be transferred to the Object Query Browser by clicking the *Import Key & Intersect Active Collection* button (see Section 5.4.2.1). The spatial selection will in this way be intersected with the active collection (the active collection should be a copy of the *Pipes* collection). The SDSS Operator can then fill the year of installation attribute of the resulting *Pipes* sub-collection with the inauguration year figure, using the standard data editing and fill operations accessible from the Pseudo-Collection sub-GUI (see Section 5.4.3.1). The SDSS Operator should do this spatial selection of pipes and the transfer operation for each residential area. Various *Pipes* sub-collections will in this way be created containing the pipes within each of the residential areas. Pipes crossing the border of a residential area will also be selected by the system. Clipping of pipes at the border will change the pipe network model and is therefore not supported by the system. Only the object identifier, the *userlinknr* (i.e. the user link number) and year of installation attributes of the objects in these sub-collections are of interest; the other attributes can be dropped.

The SDSS Analyst can then select a standard operation to unify (see Section 5.3.5) all the sub-collections that resulted from the spatial selection into one *Pipes_Res_Area* collection (see Figure 6.2).

An association between the *Pipes* collection and the *Pipe_Res_Area* collection should then be established by applying the standard navigational tool for building simple associations (see Section 5.3.2) and specifying the *userlinknr* attribute for both association keys. The association will be a specialised *one-to-many* association with multiplicity (1-----0..2) (see Section 3.2.2), since a pipe at the border regions can cross over two residential areas (which may also have different inauguration years).

The SDSS Analyst can then calculate the average year of installation for each pipe in the *Pipes* collection by selecting one of the statistics operations (viz. *obj_Calc4)* from the Pseudo-Collection sub-GUI (see Section 5.4.3.1). The operation uses the *average* statistical function (see Section 5.3.8) to calculate the average value of the associated installation year attributes of the pipe and then assigns this calculated average value to the installation year attribute of the pipe in the *Pipes* collection. The resulting average value will in most cases simply be equal to the inauguration year of the residential area in which the pipe is located. If, however, the pipe crosses a border between two residential areas, the operation will calculate the average year of inauguration for the two residential areas and assign it to the pipe installation year.

After having estimated the average year of pipe installation for each pipe in the *Pipes* collection, the SDSS Analyst can then select from the Pseudo-Collection sub-GUI the special *obj_Check3* function (which will be discussed in detail in Section 6.2.1.2), to establish an *old* pipes sub-collection.

Finally, the SDSS Analyst can access the Thematic GUI (see Section 5.4.6) to prepare the thematic maps for the Public Works Administrator, depicting the areas where *new* (<25 years) and *old* pipes (≥25 years) can be located.

The model can also be tested and further calibrated (see Sections 5.3.9 and 7.2) to ascertain that the *old* pipes classification (≥25 years) is correctly chosen so that almost all of the recorded pipe breaks have occurred in the *old* pipes category and only very few in the *new* pipes category.

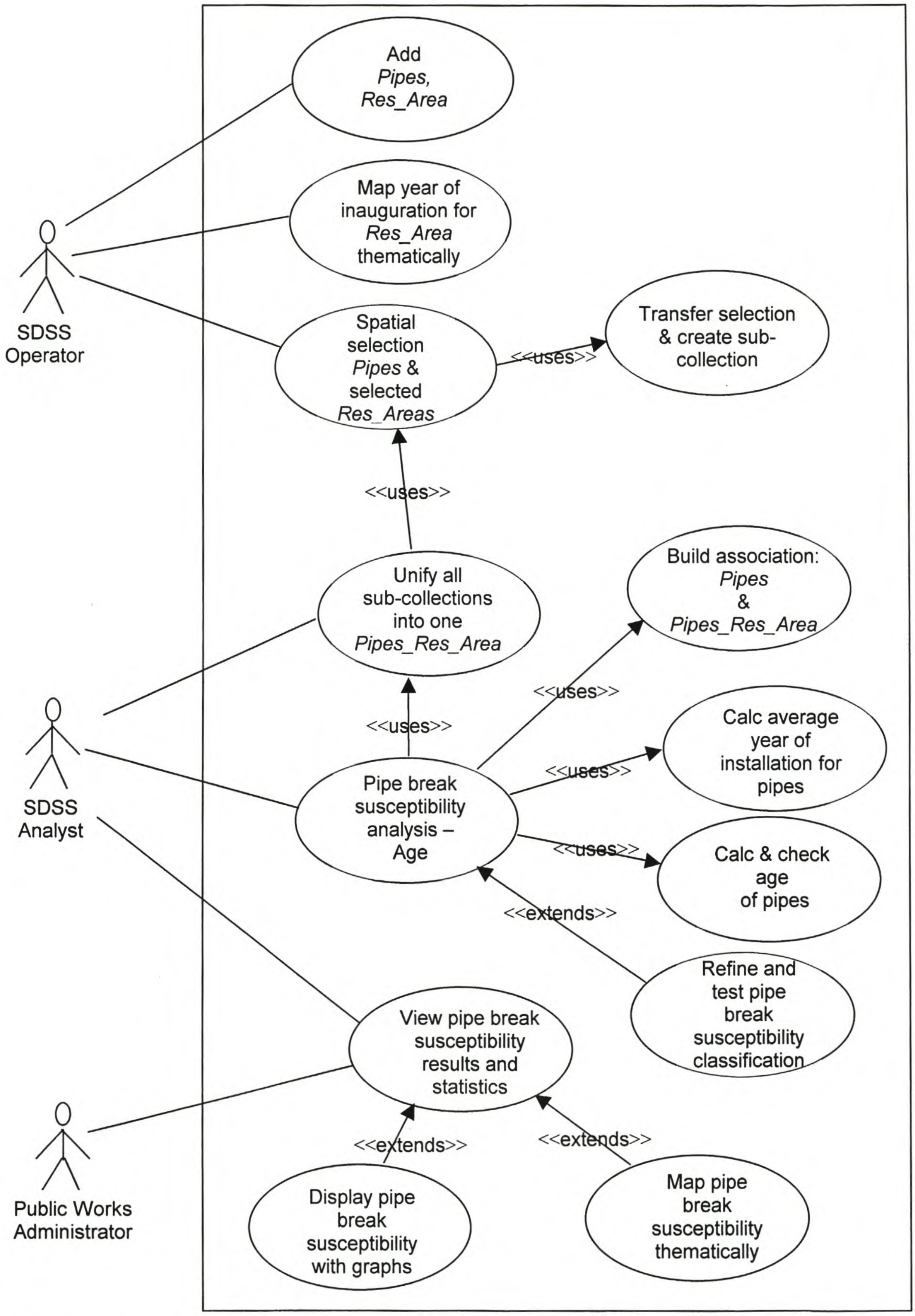

Add *Pipes*, *Res_Area*
Map year of inauguration for *Res_Area* thematically
SDSS Operator
Spatial selection *Pipes* & selected *Res_Areas*
<<uses>>
Transfer selection & create sub-collection
<<uses>>
Unify all sub-collections into one *Pipes_Res_Area*
Build association: *Pipes* & *Pipes_Res_Area*
<<uses>>
<<uses>>
SDSS Analyst
Pipe break susceptibility analysis – Age
Calc average year of installation for pipes
<<uses>>
<<uses>>
Calc & check age of pipes
<<extends>>
Refine and test pipe break susceptibility classification
View pipe break susceptibility results and statistics
Public Works Administrator
<<extends>>
<<extends>>
Display pipe break susceptibility with graphs
Map pipe break susceptibility thematically

FIGURE 6.2 Use case diagram: Pipe break susceptibility analysis – Age

Use case description: **<<Add *Pipes, Res_Areas* collection>>**

Actor Action	System Response
1. SDSS Operator selects an existing *Pipes* collection to be loaded in.	2. Adds the selected *Pipes* collection to the system.
3. SDSS Operator selects an existing *Res_Areas* collection to be loaded in.	4. Adds the selected *Res_Areas* collection to the system.

Use case description: **<<Map year of inauguration for *Res_Area* thematically>>**

Actor Action	System Response
1. SDSS Operator specifies in the Thematic GUI the inauguration year attribute of the *Res_Area* collection to be mapped thematically.	2. Transfer attribute data to ArcView where a thematic map, depicting the spatial distribution of the inauguration year of the residential areas, can be created.

Use case description: **<<Spatial selection: *Pipes* & selected *Res_Areas*>>**

Actor Action	System Response
1. The SDSS Operator uses the ArcView *Select by Theme* spatial tool to automatically select the pipes (of the *Pipes* theme) that are within a selected residential area (of the Res_Area theme). The *intersect* ArcView operator should be specified for the theme-by-theme selection.	2. Selects all the pipes within (i.e. intersect) the selected residential area.
3. Clicks the *Import Key & Intersect Active Collection* button of the Spatial Analysis sub-GUI to transfer the spatial selection to the Object Query Browser.	4. Intersects the spatial selection with the active *Pipes* collection in the Query Browser to create a *Pipes* sub-collection.
5. Selects the *obj_Calc1* operation from the Pseudo-Collection sub-GUI to update the installation year attribute of the sub-collection with the inauguration year of the residential area.	6. Automatically updates (fills) the installation year attribute of each pipe object in the sub-collection with the inauguration year figure.
7. The SDSS Operator should carry out the spatial selection for all residential areas (i.e. repeat steps 1, 3 and 5 for all residential areas).	8. Establishes the corresponding *Pipe* sub-collections according to steps 2, 4 and 6.

FIGURE 6.3(a) Use case diagram descriptions: Pipe break susceptibility analysis – Age (Part I)

Use case description: **<<Unify all sub-collections into one Pipes_Res_Area>>**

Actor Action	System Response
1 The SDSS Analyst selects the *tree_Calc3* operation from the Tree sub-GUI and specifies the sub-collections to be unified.	2. All the sub-collections that resulted from the spatial selection are unified into one *Pipes_Res_Area* collection.

Use case description: **<<Pipe break susceptibility analysis – Age>>**

Actor Action	System Response
1. The SDSS Analyst clicks the standard *Build Asso* navigation tool button and specifies the *Pipes* collection as the main collection and the *Pipes_Res_Area* as the associated collection. The *userlinknr* attribute is specified for both association keys.	2. Builds a *one*-to-*many* association between *Pipes* and *Pipes_Res_Area* via the *buildAsso* operation (that is invoked by the click event of the navigation tool button).
3. Selects the *obj_Calc4* operation from the Pseudo-Collection sub GUI.	4 Calculates the average pipe installation year for each pipe.
5. Applies the *obj_Check3* function.	6. Calculates the age of each pipe and reduces the *Pipes* collection to contain only *old* pipes with age ≥ 25 years.
Alternative Courses: After Step 6, the SDSS Analyst can now build a *one*-to-*many* association between the reduced *Pipes* collection (i.e. the *old* pipes) and the *Breaks* collection (real pipe break occurrences, see Section 7.1.3) and then apply the *collec_Calc2* operation to test and further calibrate the model. If necessary, the classification (pipe age ≥25 years) should be adjusted to obtain a better correlation between model results and real pipe break occurrences.	

Use case description: **<<View pipe break susceptibility results & stats.>>**

Actor Action	System Response
1. The SDSS Analyst specifies the thematic map settings in the *Thematic GUI* of the SDSS.	2. Transfer attribute data to ArcView where a thematic map, depicting the *old* and *new* pipes, can be created.
2. The Publics Works Administrator can then view these maps to assist him in the decision-making process regarding maintenance planning.	

FIGURE 6.3(b) Use case diagram descriptions: Pipe break susceptibility analysis – Age (Part II)

6.2.1.2 Design

The *obj_Check3* function, used for calculating the current age of the pipes and to establish an *old* pipes sub-collection, is one of the pseudo-collection level functions that can be selected from the Attributes GUI. The activity diagram (see Figure 6.4) describes the internal flow of the function. The function is implemented in the system with the following signature:

TreeOperations::obj_Check3 (treedatarow:Treebranchdatarow, propnames:Collection, datarowcollec:Treebranchdatacollec, cpropnames:Collection): Boolean

As in all *obj_Check* functions, the iteration through the collection is handled externally by operation *checkObjrows* (see Section 5.4.3.1). The *checkObjrows* operation iterates through the *Pipes* collection and for each pipe in the collection the current pipe information (*treedatarow:Treebranchdatarow*) and its associated information, if any, viz. (*datarowcollec:Treebranchdatacollec*), are passed on as parameters via the *checkTreebranchrow* function to the *obj_Check3* function. The *obj_Check3* function then performs its calculations and checks (see Figure 6.4) on the data received. The *checkTreebranchrow* function finally receives the Boolean result of the *obj_Check3* function call and will transfer the Boolean result to the *checkObjrows* operation, which will update the *Pipes* collection accordingly (i.e. remove the pipe from the collection, if *obj_Check3* is false).

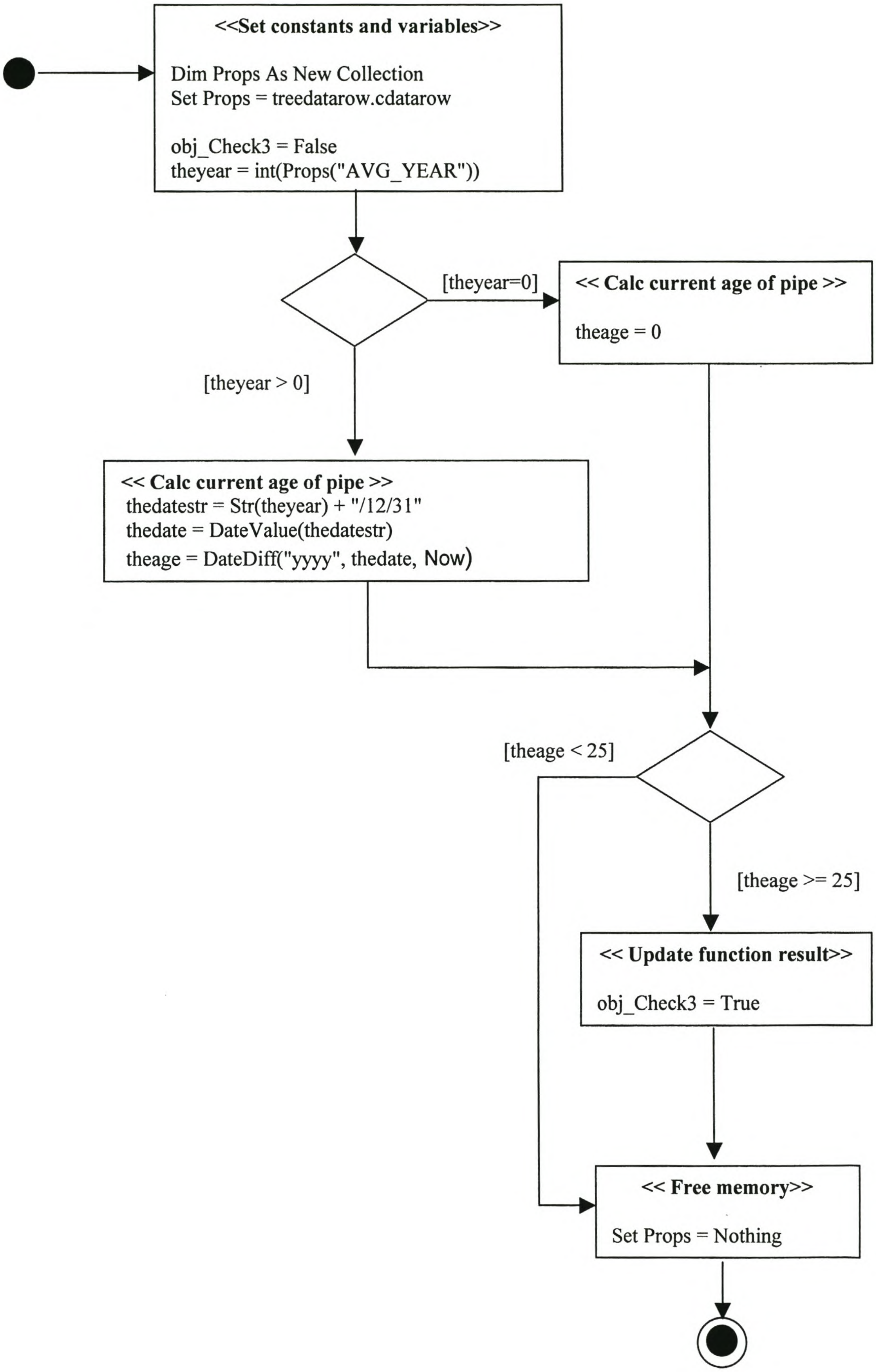

<<Set constants and variables>>
Dim Props As New Collection
Set Props = treedatarow.cdatarow
obj_Check3 = False
theyear = int(Props("AVG_YEAR"))
[theyear=0]
<< Calc current age of pipe >>
theage = 0
[theyear > 0]
<< Calc current age of pipe >>
thedatestr = Str(theyear) + "/12/31"
thedate = DateValue(thedatestr)
theage = DateDiff("yyyy", thedate, Now)
[theage < 25]
[theage >= 25]
<< Update function result>>
obj_Check3 = True
<< Free memory>>
Set Props = Nothing

FIGURE 6.4 Activity diagram of function *obj_Check3*

Activity description: **<<Set constants and variables>>**

Creates a new temporary collection *Props* and sets it equal to the collection *treedatarow.cdatarow* which holds the current pipe information.

The Boolean function *obj_Check3* is set false, which is the initial state of the function.

The average installation year of the pipe can now be read from the *Props* collection and the integer part of this figure, is stored into variable *theyear*.

Activity description: **<<Calculate current age of pipe>>**

If the average installation year equals zero (i.e. *theyear* = 0), then the pipe is not located in a residential area and the installation year can therefore not be estimated.

If the installation year was determinable, then the age of the pipe (viz. *theage*) can be determined with the VBA *DateValue* and *DateDiff* functions as shown in the activity diagram. The *DateDiff* function uses the built-in VBA function *Now* to determine the current date. The "yyyy" parameter that is used by the *DateDiff* function specifies that the function must use years as the smallest interval when calculating the age. If the pipe installation dates and subsequent replacement dates are well documented and available in digital format, then the time interval used by *DateDiff* function can be set to days in order to calculate the age more accurately. If such accurate digital data are available, then the *obj_Check3* function can be used directly without first having to estimate the installation date via the inauguration date of the containing residential area(s).

Activity description: **<< Update function result >>**

The Boolean function *obj_Check3* is set true if the age of the pipe is equal or older than 25 years (i.e. if *theage* ≥ 25).

Activity description: **<<Free memory>>**

The temporary *Props* collection is removed from memory.

FIGURE 6.5 Activity diagram descriptions for function *obj_Check3*

6.2.2 Susceptibility to breakage caused by air pockets

In this section specialised query functions and manipulation operations for pipe break susceptibility analysis with regard to air pockets (see Section 4.9), will be designed and incorporated into the SDSS.

6.2.2.1 Requirements analysis

The specialised query functions and manipulation operations of the SDSS should support the pipe break susceptibility analysis as the result of seven different types of air pocket formations in the network, viz. (i) at long pipes that also have flat slopes; (ii) at pipes with blank ends; (iii) at pipes where there are changes in diameter; (iv) at nodal high points; (v) at nodal low points; (vi) at high points along pipes; and (vii) at low points along pipes. The pipes in the network will be categorised according to the above-mentioned air-pocket formation types and then further subdivided into finer sub-types according to certain factors such as velocity, slope and diameter. The total pipe break susceptibility (due to air pockets) will then be calculated by collectively taking into account all the factors regarding air pocket formation in the pipes, using the Multifactor Evaluation Process (MFEP) that was discussed in Section 5.3.7. The collections containing the data objects and notation used to model pipe break susceptibility as the result of air pockets are depicted in Figure 6.6 (see Section 7.1.2 for data accuracy levels).

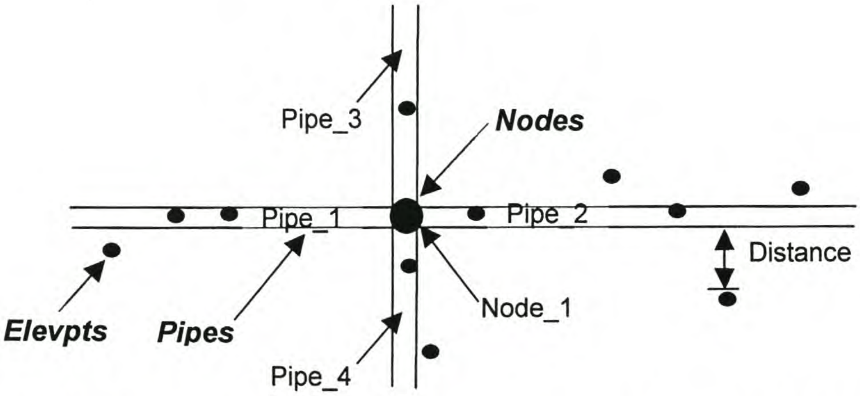


FIGURE 6.6 Description of data objects and notation used for pipe break susceptibility analysis – Air pockets

The SDSS Operator (see Figure 6.7) conducts a spatial join operation in ArcView between the *Elevpts* theme (containing the point objects with known elevation in the study area) and the *Pipes* theme. The spatial join operation finds for each *Elevpts* object the nearest *Pipes* object (see Figure 6.6). ArcView automatically creates two new temporary fields, viz. *distance* and *userlinknr,* in the ArcView *Elevpts* attribute table and copies the results of the spatial join, viz. the calculated distance to the nearest pipe (see Figure 6.6) and the *userlinknr* (i.e. the user link number) of the nearest pipe to these two new temporary fields. The SDSS Operator can then import the newly updated *Elevpts* attribute table into the Object Query Browser as described in Section 5.4.2.1 (and should drop the temporary fields that have been created in the ArcView *Elevpts* attribute table, to avoid data duplication). The *Elevpts* points that are further than 12 m away from a pipe can be excluded from the collection, using the quick query facility (see Section 5.4.3.1) and specifying the appropriate query on the *distance* attribute. These elevation points are too far away from the pipe to be used in the subsequent longitudinal slope calculations of the pipe break susceptibility analyses of Type 6 and 7 (see Figure 6.7).

A *many*-to-*many* association (see Section 3.1.4.3) must be established between the objects in the *Nodes* collection (the main collection) and objects in the associated *Pipes* collection, to be used in the Type 2, Type 3, Type 4 and Type 5 air-pocket formation checks (see Figure 6.7). The multiplicity of the association (see Section 3.2.2) is important for subsequent modelling purposes and is as follows: (2 ------1...*). This implies that each node can thus have one or more connected pipes and a pipe will always have two nodes, viz. a begin node and an end node. A node can have a minimum of one connected pipe, which will be the case when the node represents a blank end. The establishment of this specialised *many*-to-*many* association is somewhat complex, because a pipe is to be associated with a specific node if the *userbegnr* (i.e. user begin number) attribute or the *userendnr* (i.e. user end number) attribute of the pipe is the same as the *usernodenr* (i.e. user node number) of the node. The association can be established by applying the special operation for building complex associations (see Section 5.3.3).

The simpler *one*-to-*many* association between the objects in *Pipes* and the objects in *Elevpts* can be established by applying the standard navigational tool for building simple associations (see Section 5.3.2). The *userlinknr* attribute should be specified for both association keys. The *userlinknr* attribute values in the *Elevpts* collection stem from the spatial join between *Elevpts* and *Pipes* that was carried out in ArcView by the SDSS Operator (see Figure 6.7) This *one-to-many* association between *Pipes* and *Elevpts* will then be used in the pipe break susceptibility analyses of air-pocket formation Types 6 and 7 (see Figure 6.7).

The pipe break susceptibility analyses for each of the seven types of air pocket formation can then be carried out by the SDSS Analyst (see Figure 6.7). The detailed classification of the sub-types are given in Table 7.1 of Chapter 7, where the SDSS is applied to the water distribution system of a study area. The pipes can be classified according to the schema shown in Table 7.1 by applying the quick query facility (see Section 5.4.3.1) for the simpler queries and the function *obj_Check2*, which can be selected from the Pseudo-Collection sub-GUI (see Figure 5.4.3.1), for the more complex queries. A complex query can have a query domain that includes an associated collection and the query can also include derived values. The *obj_Check2* function will be discussed in detail in Section 6.2.2.2.

The SDSS Analyst should expand the resultant *Nodes* collections from the Types 2, 3, 4 and 5 pipe break susceptibility analyses, with the *Expand* operation (see Section 5.3.4) so that the associated pipes can be separated from the nodes and exported as simple pipe collections.

The SDSS Analyst can summarise the results obtained from the individual pipe break susceptibility analyses of Type 1-7, by using the MFEP operation (see Section 5.3.7) to calculate a *Total_airbreak* figure for each pipe in the *Pipes* collection. The recommended *factor weights* (see Section 5.3.7) that should be assigned to the sub-collection resulting from the Type 1-7 analyses are shown in Table 7.1.

Finally, the SDSS Analyst can access the Graphs GUI (see Section 5.4.5) and the Thematic GUI (see Section 5.4.6) to prepare the graphs and thematic maps for the Public Works Administrator. The Graph subsystem of the SDSS can also be used to compile a cumulative distribution graph (see Section 5.4.5) of the sorted total pipe break susceptibility figures (i.e. the sorted *Total_airbreak* figures). This graph can then be examined visually by the SDSS Analyst to determine a suitable upper-class interval for pipes with *high* pipe break susceptibility (due to air pockets). This classification can then further be refined to show pipes with *very high* and *extremely high* break susceptibility (see Section 7.2).

The model can also be tested and further calibrated (see Sections 5.3.9 and 7.2), if there is data available on airburst occurrences in the pipe network.

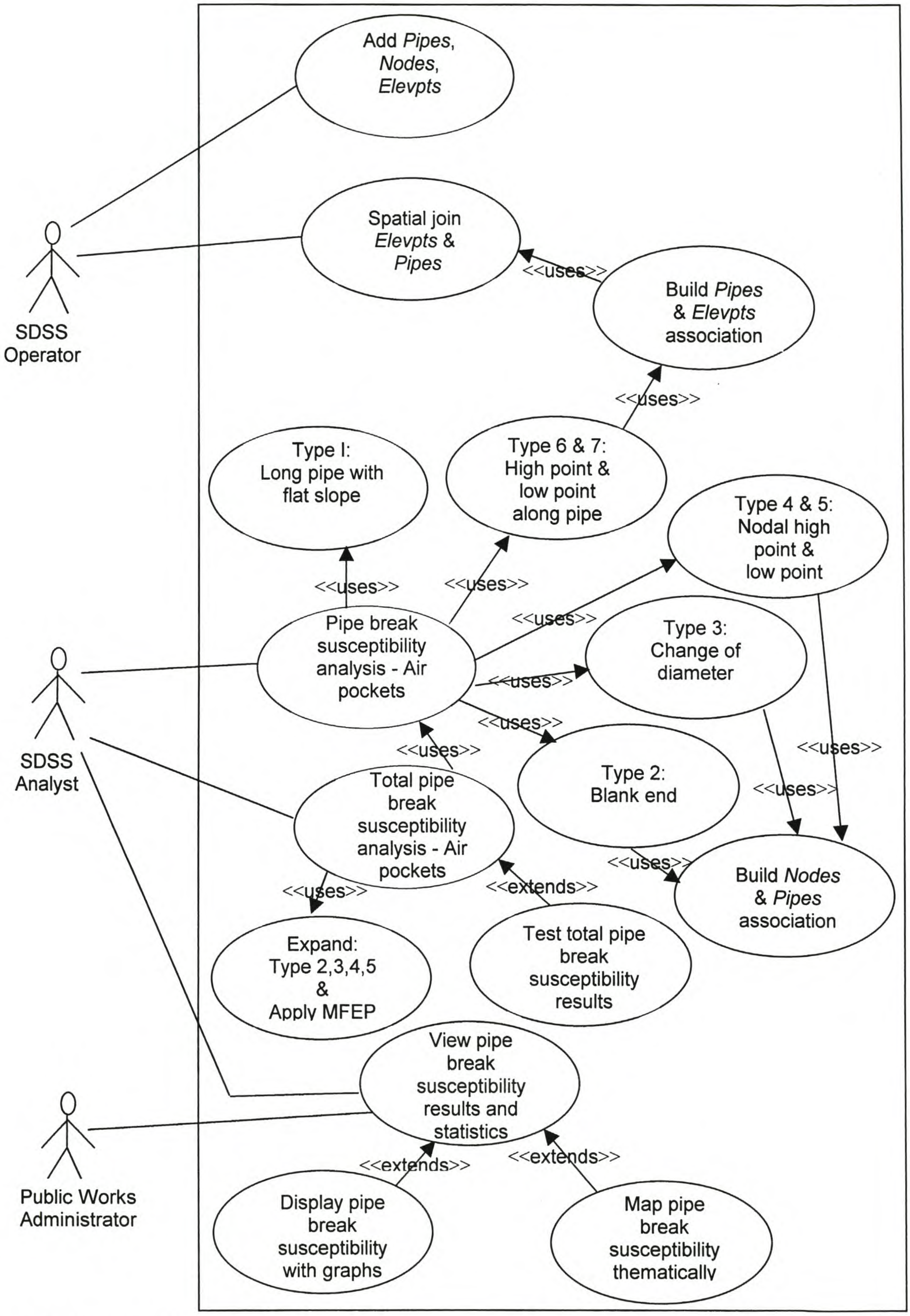

Add *Pipes*, *Nodes*, *Elevpts*
Spatial join *Elevpts* & *Pipes*
<<uses>>
Build *Pipes* & *Elevpts* association
SDSS Operator
<<uses>>
Type I: Long pipe with flat slope
Type 6 & 7: High point & low point along pipe
Type 4 & 5: Nodal high point & low point
<<uses>>
<<uses>>
<<uses>>
Pipe break susceptibility analysis - Air pockets
Type 3: Change of diameter
<<uses>>
<<uses>>
SDSS Analyst
<<uses>>
Type 2: Blank end
<<uses>>
<<uses>>
Total pipe break susceptibility analysis - Air pockets
<<uses>>
Build *Nodes* & *Pipes* association
<<uses>>
<<extends>>
Expand: Type 2,3,4,5 & Apply MFEP
Test total pipe break susceptibility results
View pipe break susceptibility results and statistics
Public Works Administrator
<<extends>>
<<extends>>
Display pipe break susceptibility with graphs
Map pipe break susceptibility thematically

FIGURE 6.7 Use case diagram: Pipe break susceptibility analysis – Air pockets

Use case description: **<<Add *Pipes, Nodes, and Elevpts* collection>>**

Actor Action	System Response
1. SDSS Operator selects an existing *Pipes* collection to be loaded in.	2. Adds the selected *Pipes* collection to the system.
3. SDSS Operator selects an existing *Nodes* collection to be loaded in.	4. Adds the selected *Nodes* collection to the system.
5. SDSS Operator selects an existing *Elevpts* collection to be loaded in.	6. Adds the selected *Elevpts* collection to the system.

Use case description: **<<Spatial join Elevpts & Pipes>>**

Actor Action	System Response
1. The SDSS Operator applies the ArcView spatial join operation and specifies the two themes: *Elevpts* (main theme) and *Pipes* (associated theme) to be joined spatially.	2. Builds association: *Elevpts* object with its nearest *Pipes* object and creates a *distance* field where the distance to the nearest pipe is written. A *userlinknr* field is also created and updated accordingly.
3. The SDSS Operator clicks the customised export button in ArcView to export the *Elevpts* attribute table. The SDSS Operator then removes (drops) the *distance* and *userlinknr* fields.	4. The *Elevpts* attribute table is exported in the query result collection file format (see Appendix A). *Distance* and *userlinknr* fields in the ArcView *Elevpts* attribute table are dropped.
5. The SDSS Operator clicks the *Import* navigation button in the Object Query Browser to import the new *Elevpts* collection.	6. The new updated *Elevpts* collection, which contains the *distance* and *userlinknr* attributes, is added to the Object Query Browser project.
7. The SDSS Operator applies the quick query facility and specifies the following query: *distance* ≤12	8. Reduces the *Elevpts* collection so that it now contains only points that are within 12 m from a pipe.

FIGURE 6.8(a) Use case diagram descriptions: Pipe break susceptibility Analysis – Air pockets (Part I)

Use case description: **<<Pipe break susceptibility analysis – Air Pockets>>**

Actor Action	System Response
1. The SDSS Analyst applies the quick query facility and specifies appropriate short consecutive queries on the *Pipes* collection, which are based on the classification schema for Type 1 air pockets as given in Table 7.1 of Chapter 7. The query on pipe slope (see Table 7.1) can also be done with the quick query facility, provided the pipe slope is stored as an attribute along with the other pipe attributes. The model used in the SDSS, however, stores only the upstream and downstream elevations of each pipe, since the pipe slope can be derived from these elevations and the pipe length. The query on the pipe slope is thus a complex query that should be executed with the *obj_Check2* function.	2. Reduces the *Pipes* collection so that it contains only pipes that fall within the classification given in Table 7.1 for Type 1 air pocket formation.
3. The SDSS Analyst applies the *tree_Calc1* operation to build the complex association between the *Nodes* collection and the *Pipes* collection. The SDSS Analyst then uses the quick query facility to select all nodes that have only one associated pipe (query statement: *Npipes* = 1). He then applies the *obj_Check2* function to query the associated pipes of the *Nodes* collection. The query statements are set up according to the classification schema for Type 2 air pockets as given in Table 7.1	4. Builds a *many*-to-*many* association between *Nodes* and *Pipes.* Reduces the *Nodes* collection so that it contains only blank ends, i.e. nodes that have only one associated pipe. Further reduces the *Nodes* collection so that it contains only associated pipes that fall within the classification given in Table 7.1 for Type 2 air pocket formation.

FIGURE 6.8(b) Use case diagram descriptions: Pipe break susceptibility analysis – Air pockets (Part II)

5. The SDSS Analyst applies the *obj_Check2* function to query the associated pipes of the each node in the *Nodes* collection. The query statements are set up according to the classification schema for Type 3 air pocket formation as given in Table 7.1. The classification of the last two Type 3 subcategories shown in Table 7.1, viz. medium and large transition in diameter, can be obtained with the *quick query* facility and specifying the query statement *Npipes=2*. In this way the results obtained from the *obj_Check2* function, will be further reduced so that only the nodes with two pipes associated (e.g. a larger diameter pipe that changes to a smaller diameter pipe) will be listed.	6. Reduces the *Nodes* collection so that it contains only associated pipes that fall within the classification given in Table 7.1 for Type 3 air pocket formation.
7. The SDSS Analyst applies the *quick query* facility and specifies the following query: *Npipes* >1. In this way the nodes will be selected with more than one pipe associated to it. The blank end pipes that were already analysed in the previous use case, will now be excluded from the collection. The *obj_Check2* function can now be applied to find all nodes that are nodal low points. The classification of the last two Type 4 subcategories (see Table 7.1) can be obtained by applying the *tree_Calc2* operation (the expand operation) on the already reduced *Nodes* collection. The final classification can then be obtained by specifying quick queries on the pipes (of the expanded nodes).	8. Reduces the *Nodes* collection so that it contains only associated pipes that fall within the classification given in Table 7.1 for Type 4 air pocket formation.
9. The SDSS Analyst uses the Object Query Browser in very much the same way as described for Type 4, but the queries should be specified for Type 5 air-pocket formation checks.	10. Reduces the *Nodes* collection so that it contains only associated pipes that fall within the classification given in Table 7.1 for Type 5 air pocket formation.

FIGURE 6.8(c) Use case diagram descriptions: Pipe break susceptibility analysis – Air pockets (Part III)

Actor Action	System Response
11. The SDSS Analyst applies the navigation tool for building simple associations and specifies the *Pipes* collection as the main collection and the *Elevpts* as the associated collection (*userlinknr* is specified for both association keys). He then applies the *obj_Check2* function to query the associated elevation points of the pipes in the *Pipes* collection. The classification of the last two subcategories of Type 6 shown in Table 7.1 can be obtained by applying the quick query facility and specify the appropriate simple query statements to obtain the final classification results.	12. Builds a *one*-to-*many* association between *Pipes* and *Elevpts.* Reduces the *Pipes* collection so that it contains only pipes that fall within the classification given in Table 7.1 for Type 6 air pocket formation.
13. The SDSS Analyst uses the Object Query Browser in very much the same way as described for Type 6, but the queries should be specified for Type 7 air-pocket formation checks.	14. Reduces the *Pipes* collection so that it contains only pipes that fall within the classification given in Table 7.1 for Type 7 air pocket formation.

FIGURE 6.8(d) Use case diagram descriptions: Pipe break susceptibility analysis – Air pockets (Part IV)

Use case description: **<<Total pipe break susceptibility analysis – Air Pockets>>**

Actor Action	System Response
1. SDSS Analyst applies the *tree_Calc2* operation to expand the node collections resulting from the pipe break susceptibility analyses for Types 2,3,4 and 5.	2. Expands the specified node collections.
3. The SDSS Analyst applies *MFEP* operation and specifies *Pipes* as the target collection. He then specifies the sub-collection containing the pipes from an individual pipe break susceptibility analysis of an air pocket formation type. The *factor weight* must also be specified.	4. For each pipe in *Pipes*, check if it is contained in the sub-collection. If so, then the specified *factor weight* is added to the *Total_airbreak* attribute of the current pipe in the *Pipes* collection.
5. Repeat steps 1 to 3 for each sub-collection resulting from the Type1-7 analyses.	6. Continues with the MFEP process.
Alternative Courses: After step 6, the SDSS Analyst can now also build a *one*-to-*many* association between the *Pipes* collection and the *Breaks_air* collection (real airburst occurrences in the network, see Section 7.1.3) and then apply the *collec_Calc2* operation to test and further calibrate the model.	

Use case description: **<<View pipe break susceptibility results and statistics>>**

Actor Action	System Response
1. The SDSS Analyst exports the pipe break susceptibility figures to the *Graph* subsystem and sorts them there in ascending order.	2. The cumulative pipe break susceptibility distribution graph is shown. This graph may also be of interest to the Public Works Administrator.
3. The SDSS Analyst can group the pipes into different *high* pipe break susceptibility classes and can then export these sub-collections.	4. The *high* pipe break susceptibility collection(s) are exported and are thus made persistent.
Alternative Courses: After Step 4, the SDSS Analyst can prepare thematic maps showing the pipes with *high* break susceptibility (as the result of air pockets). The Publics Works Administrator can then view these maps to assist him in the decision-making process regarding maintenance planning.	

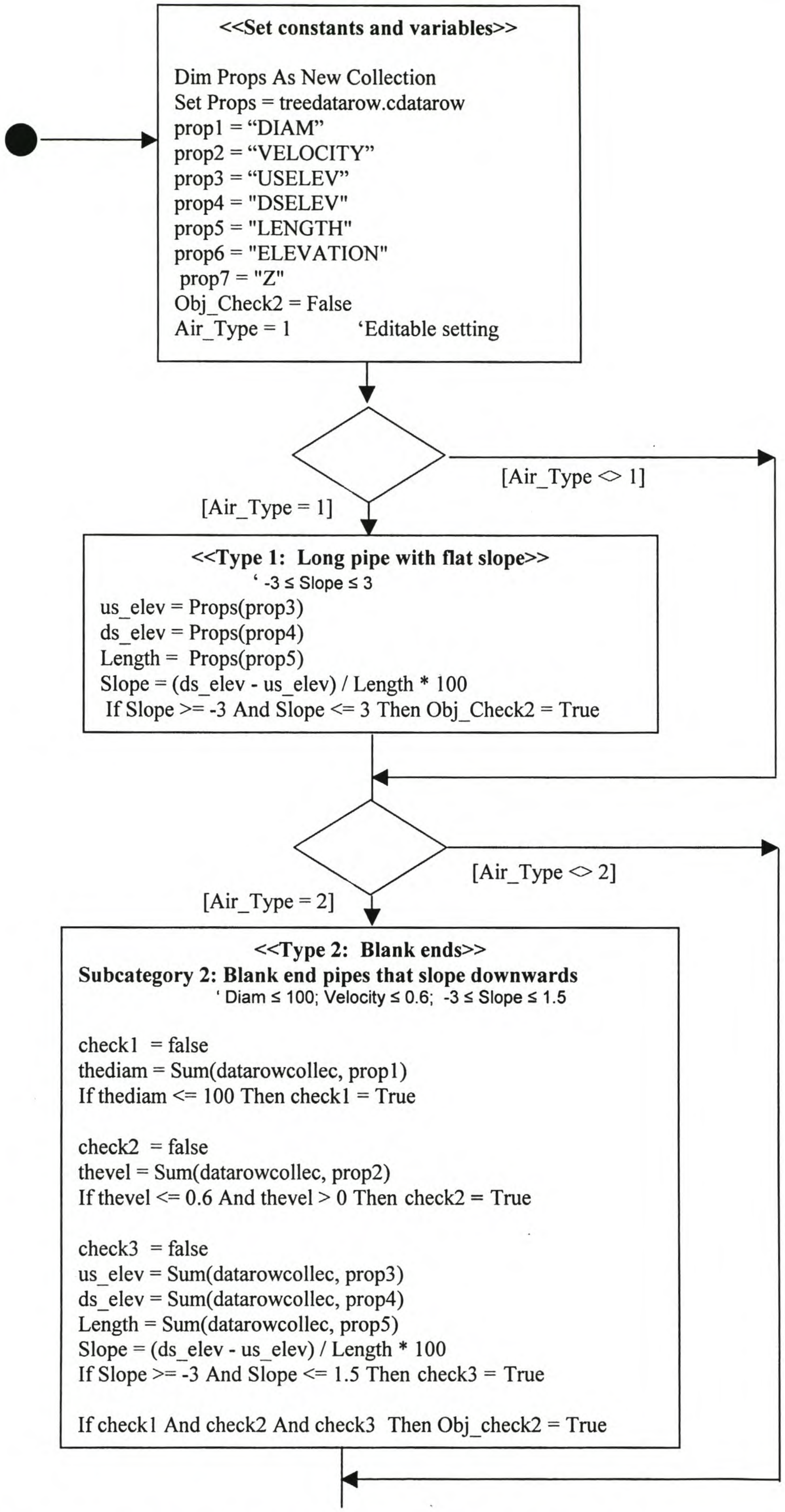
FIGURE 6.8(e) Use case diagram descriptions: Pipe break susceptibility analysis – Air pockets (Part V)

6.2.2.2 Design

Function *obj_Check2* contains the complex query functionality (the simpler queries are managed by the quick query facility) needed to analyse and assess the air-pocket formation potential in the network, as described in the previous section. The activity diagram for the *obj_Check2* function is shown in Figure 6.9, describing the internal flow of the function. The function is implemented in the system with the following signature:

TreeOperations::obj_Check2 (treedatarow:Treebranchdatarow, propnames:Collection, datarowcollec:Treebranchdatacollec, cpropnames:Collection): Boolean

As in all *obj_Check* functions, the iteration through the collection is handled externally by operation *checkObjrows* (see Section 5.4.3.1). The *checkObjrows* operation iterates through the objects in the collection (which will be *Pipes* for Type 1, 6 and 7 air-pocket formation checks and *Nodes* for the air-pocket formation checks of Type 2, 3, 4 and 5) and for each object in the collection the current object information (*treedatarow:Treebranchdatarow*) and its associated object information, if any, viz. (*datarowcollec:Treebranchdatacollec*), are passed on as parameters via the *checkTreebranchrow* function to the *obj_Check2* function. The *obj_Check2* function then performs its calculations and checks (see Figure 6.9) on the data received. The *checkTreebranchrow* function finally receives the Boolean result of the *obj_Check2* function call and will transfer the Boolean result to the *checkObjrows* operation, which will update the *Pipes/Nodes* collection accordingly (i.e. remove the pipe/node from the collection, if *obj_Check2* is false).



<<Set constants and variables>>

```
Dim Props As New Collection
Set Props = treedatarow.cdatarow
prop1 = “DIAM”
prop2 = “VELOCITY”
prop3 = “USELEV”
prop4 = "DSELEV"
prop5 = "LENGTH"
prop6 = "ELEVATION"
 prop7 = "Z"
Obj_Check2 = False
Air_Type = 1          ‘Editable setting
```

<<Type 1: Long pipe with flat slope>>

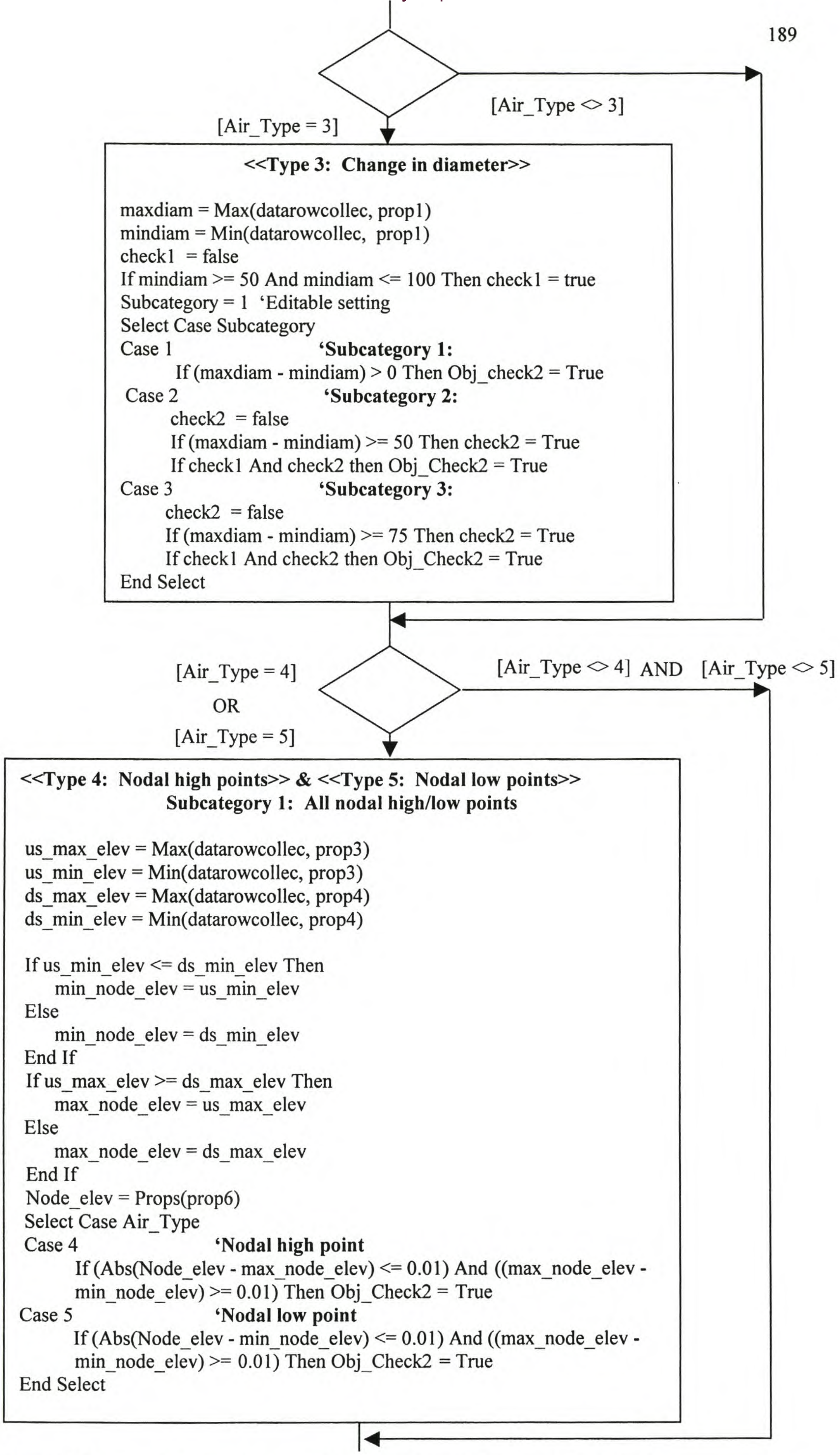
```
‘ -3 ≤ Slope ≤ 3
us_elev = Props(prop3)
ds_elev = Props(prop4)
Length =  Props(prop5)
Slope = (ds_elev - us_elev) / Length * 100
 If Slope >= -3 And Slope <= 3 Then Obj_Check2 = True
```

<<Type 2: Blank ends>>

Subcategory 2: Blank end pipes that slope downwards

```
‘ Diam ≤ 100; Velocity ≤ 0.6;  -3 ≤ Slope ≤ 1.5

check1  = false
thediam = Sum(datarowcollec, prop1)
If thediam <= 100 Then check1 = True

check2  = false
thevel = Sum(datarowcollec, prop2)
If thevel <= 0.6 And thevel > 0 Then check2 = True

check3  = false
us_elev = Sum(datarowcollec, prop3)
ds_elev = Sum(datarowcollec, prop4)
Length = Sum(datarowcollec, prop5)
Slope = (ds_elev - us_elev) / Length * 100
If Slope >= -3 And Slope <= 1.5 Then check3 = True

If check1 And check2 And check3  Then Obj_check2 = True
```

FIGURE 6.9(a) Activity diagram for function *obj_Check2* (Part I)



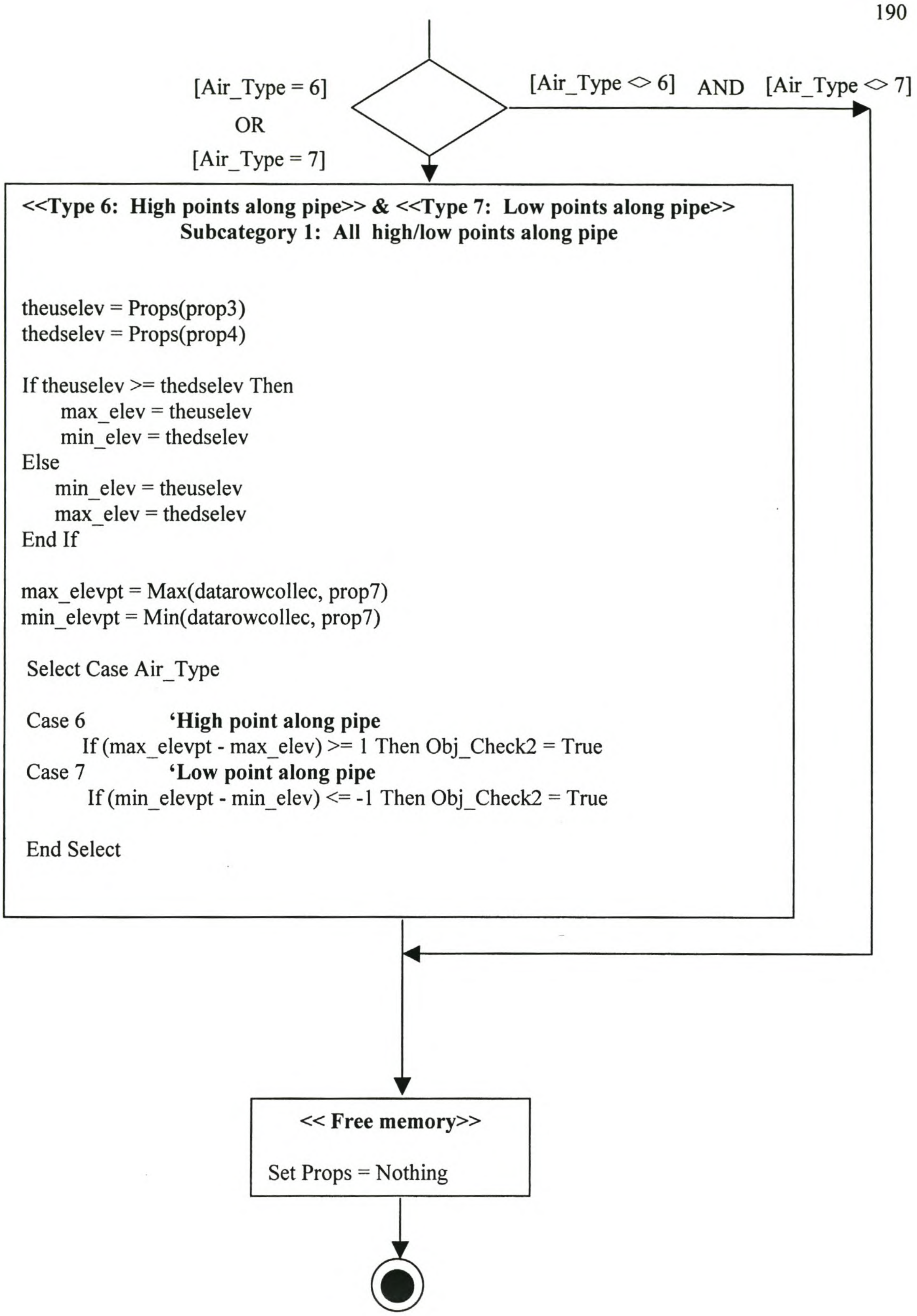
<<Type 3: Change in diameter>>

```
maxdiam = Max(datarowcollec, prop1)
mindiam = Min(datarowcollec, prop1)
check1 = false
If mindiam >= 50 And mindiam <= 100 Then check1 = true
Subcategory = 1 'Editable setting
Select Case Subcategory
Case 1                        'Subcategory 1:
     If (maxdiam - mindiam) > 0 Then Obj_check2 = True
Case 2                        'Subcategory 2:
     check2 = false
     If (maxdiam - mindiam) >= 50 Then check2 = True
     If check1 And check2 then Obj_Check2 = True
Case 3                        'Subcategory 3:
     check2 = false
     If (maxdiam - mindiam) >= 75 Then check2 = True
     If check1 And check2 then Obj_Check2 = True
End Select
```

<<Type 4: Nodal high points>> & <<Type 5: Nodal low points>>
Subcategory 1: All nodal high/low points

```
us_max_elev = Max(datarowcollec, prop3)
us_min_elev = Min(datarowcollec, prop3)
ds_max_elev = Max(datarowcollec, prop4)
ds_min_elev = Min(datarowcollec, prop4)

If us_min_elev <= ds_min_elev Then
   min_node_elev = us_min_elev
Else
   min_node_elev = ds_min_elev
End If
If us_max_elev >= ds_max_elev Then
   max_node_elev = us_max_elev
Else
   max_node_elev = ds_max_elev
End If
Node_elev = Props(prop6)
Select Case Air_Type
Case 4                    'Nodal high point
     If (Abs(Node_elev - max_node_elev) <= 0.01) And ((max_node_elev -
     min_node_elev) >= 0.01) Then Obj_Check2 = True
Case 5                    'Nodal low point
     If (Abs(Node_elev - min_node_elev) <= 0.01) And ((max_node_elev -
     min_node_elev) >= 0.01) Then Obj_Check2 = True
End Select
```

FIGURE 6.9(b) Activity diagram for function *obj_Check2* (Part II)



<<Type 6: High points along pipe>> & <<Type 7: Low points along pipe>>
Subcategory 1: All high/low points along pipe

```
theuselev = Props(prop3)
thedselev = Props(prop4)

If theuselev >= thedselev Then
      max_elev = theuselev
      min_elev = thedselev
Else
     min_elev = theuselev
     max_elev = thedselev
End If

max_elevpt = Max(datarowcollec, prop7)
min_elevpt = Min(datarowcollec, prop7)

 Select Case Air_Type

 Case 6              'High point along pipe
          If (max_elevpt - max_elev) >= 1 Then Obj_Check2 = True
 Case 7              'Low point along pipe
           If (min_elevpt - min_elev) <= -1 Then Obj_Check2 = True

 End Select
```

<< Free memory>>

```
Set Props = Nothing
```

FIGURE 6.9(c) Activity diagram for function *obj_Check2* (Part III)

Activity description: **<<Set constants and variables>>**

Creates a new temporary collection *Props* and sets it equal to the collection *treedatarow.cdatarow* which holds the current object information.

All attribute names used in the operation are stored into variables *Prop1..7*. The Boolean function *obj_Check2* is set false, which is the initial state of the function.

Air-pocket formation type (1..7) to be analysed, can be specified in an dialogue box which will update the *Air_Type* variable.

Activity description: **<<Type 1: Long pipe with flat slope>>**

The upstream and downstream elevations of each pipe, as well as the pipe length, are read from the *Props* collection and stored into appropriate variables. The slope, which is a function of these three variables, is then calculated. The Boolean function *obj_Check2* is set true if the slope of the pipe is within the specified criteria.

Activity description: **<<Type 2: Blank ends>>**

At blank ends the node objects in the *Nodes* collection have only one associated pipe object from the associated *Pipes* collection. The sum of the values of an associated attribute would thus in this case be the same as retrieving the single attribute value. The *sum* function (see Section 5.3.8) can therefore be used to retrieve attribute values for diameter, velocity, length, upstream elevation and downstream elevation of the associated pipe. The slope, which is a function of the latter three variables, is then calculated. The Boolean function *obj_Check2* is set true if the diameter, velocity and slope of the pipe are within the specified criteria.

Activity description: **<<Type 3: Change in diameter>>**

The *max* function (see Section 5.3.8) is applied to retrieve for each node the maximum diameter of all its associated pipes. The *min* function (see Section 5.3.8) is applied to retrieve for each node the minimum diameter of all its associated pipes. The first query is to check if the minimum diameter is within the specified criteria. The subcategory (i.e. the sub-type) of the analysis must now be specified. In *Subcategory 1* the pipes are checked for diameter changes of any amount. In *Subcategory 2* the pipes are checked for diameter changes of 50 mm and larger, and if the minimum diameter is within the specified range. In *Subcategory 3* the pipes are checked for diameter changes of 75 mm and larger, and if the minimum diameter is within the specified range. In each subcategory, the Boolean function *obj_Check2* is set true if all the checks that where carried out in the subcategory returned *true* results.

FIGURE 6.10(a) Activity diagram descriptions for *obj_Check2* (Part I)

Activity description: **<<Type 4: Nodal high points>> & <<Type 5: Nodal low points>>**

The *max* function (see Section 5.3.8) is applied to retrieve for each node (in the *Nodes* collection), the maximum upstream elevation of all its associated pipes (in the *Pipes* collection) and store the results in the variable *us_max_elev*. The *max* function is then applied to retrieve for each node the maximum downstream elevation of all its associated pipes and store the results in the variable *ds_max_elev*. The terms 'upstream' and 'downstream' are topographically not necessarily correct, since they represent the start and end node of a pipe in the direction of flow (during peak-hour). Therefore, the maximum of the two 'upstream' and 'downstream' elevations is determined (and stored into variable *max_node_elev*) to obtain the maximum nodal elevation of the group of pipes connected to the current node being analysed. The same process can be applied to determine the minimum nodal elevation and the result is stored into variable *min_node_elev*. The current node that is being analysed can now be tested for nodal high point by comparing its nodal elevation attribute with the *max_node_elev* value. For the node to be a nodal high point its elevation should be the same as the *max_node_elevation* (a difference of 0.01 m can be allowed, to make provision for the rounding errors that may occur in real number comparison) and the pipes connected to the nodes should not have too flat slopes, since there would then not really be a high point. The same type of checks can be applied to identify nodal low points. The Boolean function *obj_Check2* is set true if all the checks confirm that the node is a nodal high point (or nodal low point, if Air_Type=5).

Activity description: **<<Type 6: High point on pipe>> & <<Type 7: Low point on pipe>>**

The start and end (or upstream and downstream) elevations are compared to find the maximum nodal elevation of each pipe – the result is stored into variable *max_elev*. The minimum nodal elevation *min_elev* of each pipe can be determined in the same way. The associated elevation points along each pipe are now queried using the *max* and *min* functions (see Section 5.3.8) to find the maximum and minimum elevation point along the pipe. The results are stored into variables *max_elevpt* and *min_elevpt*. The current pipe (in the *Pipes* collection) that is being analysed can now be tested for a high point along the pipe by comparing the *max_elevpt* with the *max_elev* value. Pipes, with high points will be identified if (*max_elevpt* - *max_elev)* ≥ 1. The 1 m cut-off makes provision for elevation measurement inaccuracy and will ascertain that only pipes with well-defined high points will be identified. Low points along the pipes can be tested in very much the same way by comparing the variables *min_elevpt* and *min_ele*v. The Boolean function *obj_Check2* is set true if all the checks confirm that the pipe has a high point (or low point, if *Air_Type =7*).

Activity description: **<<Free memory>>**

The temporary *Props* collection is removed from memory.

FIGURE 6.10(b) Activity diagram descriptions for *obj_Check2* (Part II)

6.2.3 Susceptibility to breakage caused by trees

In this section specialised query functions and manipulation operations for pipe break susceptibility analysis with regard to tree roots (see Section 4.5), will be designed and incorporated into the SDSS.

6.2.3.1 Requirements analysis

The tree size and its distance away from the pipe are the two important factors that must be taken into account when analysing pipe break susceptibility caused by trees (see Section 4.5). The collections containing the data objects and notation used for modelling the pipe break susceptibility as the result of nearby trees, taking into account the two main factors viz. tree *size* and *distance* away from nearest pipe, are depicted in Figure 6.11 (see Sections 7.1.2 and 7.1.4 for data accuracy levels).

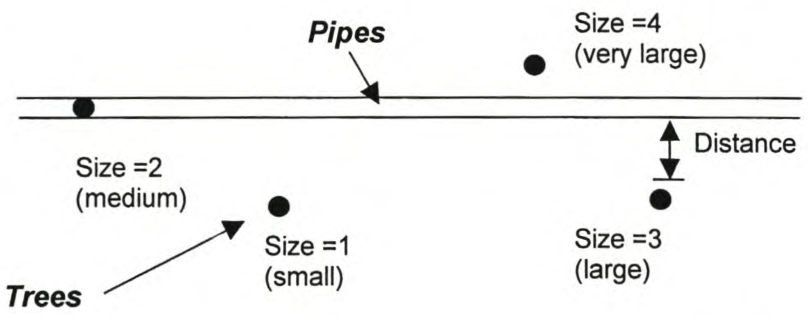


FIGURE 6.11 Description of data objects and notation used for pipe break susceptibility analysis – Trees

The tree species should in fact also be taken into account when analysing pipe breaks as the result of tree roots (see Section 4.5). This will, however, require a detailed tree survey of the study area - which would be too costly - and is therefore not included in this version of the SDSS.

The SDSS Operator (see Figure 6.12) can enter the tree object data manually or it can be captured automatically. The automatic capturing process is, however, not yet fully operational in this version of the SDSS. Both methods will eventually provide a *Trees* collection with tree objects that have a *size* attribute. With the manual method the *size* attribute is qualified on an ordinal scale using the following discrete numbers (1 = small, 2 = medium, 3 = large and 4 = very large). The scale is based on the crown diameter (<7 m = small, 7-14 m = medium, 14-21 m = large and >21 m = very large). The automatic method uses a ratio scale based on the crown area (see Figure 6.13).

The SDSS Operator conducts an ArcView spatial join operation between *Trees* and *Pipes* (see Figure 6.12). The spatial join operation finds for each *Trees* object the nearest *Pipes* object (see Figure 6.11). ArcView automatically calculates the distance to the nearest pipe, as well as the user link number of the nearest pipe and temporarily stores the data in the *Trees* attribute table. The SDSS Operator can then import the newly updated *Trees* attribute table into the Object Query Browser as described in Section 5.4.2.1.

The SDSS Analyst (see Figure 6.12) analyses the pipe-breaking influence each tree has on its nearest pipe. This pipe-breaking influence can be modelled by a fuzzy function (see Section 2.2.2) of two input fuzzy variables, viz. *size* and *distance* to nearest pipe. The fuzzy controller, which is linked to the SDSS, can be called from the Fuzzy GUI (see Section 5.4.4). The recommended input membership functions for *size* and *distance*, the output membership function *single_risk,* and also the rule base are shown in Section 7.3.3, when the SDSS will be applied to the study area. When the fuzzy controller has finished the fuzzy inference calculations, the *Trees* collection objects should be updated (see Section 5.4.4) with the corresponding fuzzy controller output values. The output value (viz. *single_risk*) represents the pipe break susceptibility as the result of a single tree.

The SDSS Analyst further calculates the accumulated pipe break susceptibility of each pipe, taking into account the pipe-breaking potential of all trees that are associated with the pipe. A *one*-to-*many* association (see Section 3.1.4.3) must be established between each pipe and its associated trees. The link between the objects can be established by applying the navigational tool for building simple associations (see Section 5.3.2) and specify *Pipes* as the main collection and *Trees* as the collection to be associated. The *userlinknr* attribute can be specified for both association keys. The *userlinknr* attribute values in the *Trees* collection stem from the spatial join between *Trees* and *Pipes* that was carried out in ArcView by the SDSS Operator (see Figure 6.12). A quick method to find the accumulated pipe break susceptibility figure for a pipe would be to simply calculate the sum of all the pipe break susceptibility figures that are stored in the associated *Trees* collection. This, however, can give misleading results, since the long pipes in the network will then almost always have a higher accumulated pipe break susceptibility figure than the short pipes in the network, because the longer the pipe is, the greater is the chance that there will be trees located alongside the pipe that can cause pipe breakage. A more realistic accumulated pipe break susceptibility figure can be obtained by applying the *obj_Calc5* operation that can be selected from the Pseudo-Collection sub-GUI (see Section 5.4.31). The *obj_Calc5* operation takes the pipe length into account when accumulating the associated *single_risk* factors; this will be discussed in detail in Section 6.2.3.2.

The SDSS Analyst can further refine the model results. The *Pipes* collection, containing the calculated accumulated pipe break susceptibility figures, may in the end be reduced so that it contains only the small diameter pipes ($d \leq 150$ mm). In this way the large diameter pipes, with the higher section modulus that can resist the bending moment caused by the tree roots (see Section 4.5), can be excluded from the *high* pipe break susceptibility classification.

The SDSS Analyst can finally prepare the statistic figures, graphs and thematic maps that are of interest to the Public Works Administrator. With the Graph subsystem a graph can be drawn of the sorted pipe break susceptibility figures, showing the cumulative pipe break susceptibility distribution (see Section 5.4.5). This graph can then be examined visually by the SDSS Analyst to determine a suitable upper-class interval for pipes with *high* pipe break susceptibility (due to trees). This classification can then further be refined to show pipes with *very high* and *extremely high* break susceptibility. The model can also be tested and calibrated (see Sections 5.3.9 and 7.2).

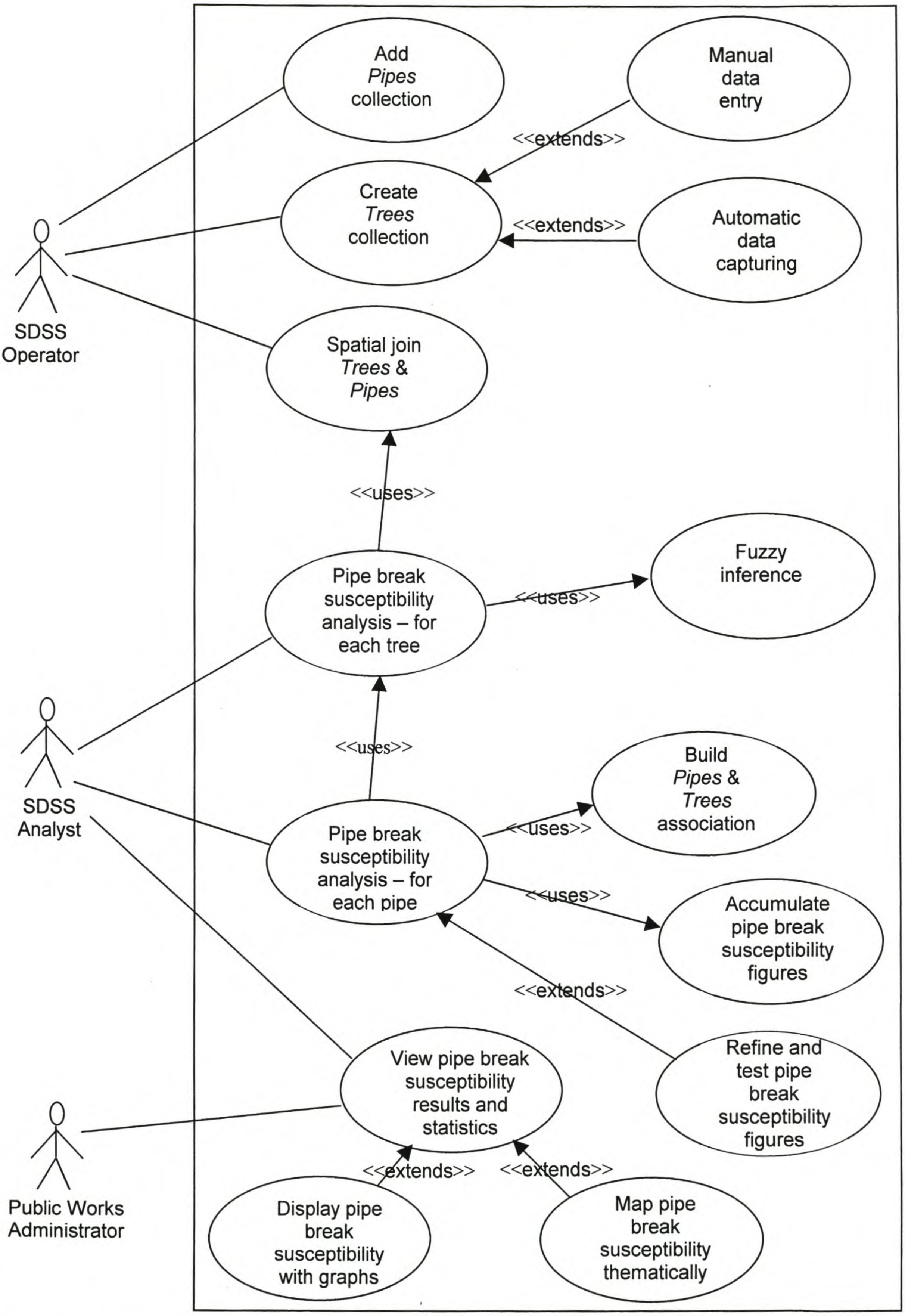

Add *Pipes* collection
Manual data entry
<<extends>>
Create *Trees* collection
<<extends>>
Automatic data capturing
SDSS Operator
Spatial join *Trees* & *Pipes*
<<uses>>
Pipe break susceptibility analysis – for each tree
<<uses>>
Fuzzy inference
<<uses>>
SDSS Analyst
Build *Pipes* & *Trees* association
<<uses>>
Pipe break susceptibility analysis – for each pipe
<<uses>>
Accumulate pipe break susceptibility figures
<<extends>>
Refine and test pipe break susceptibility figures
View pipe break susceptibility results and statistics
Public Works Administrator
<<extends>>
<<extends>>
Display pipe break susceptibility with graphs
Map pipe break susceptibility thematically

FIGURE 6.12 Use case diagram: Pipe break susceptibility analysis – Trees

Use case description: **Add *Pipes* collection**

Actor Action	System Response
1. SDSS Operator selects an existing *Pipes* collection to be loaded in.	2. Adds the selected *Pipes* collection to the system.

Use case description: **Create *Trees* collection**

Actor Action	System Response
1. SDSS Operator opens a new *Trees* theme in ArcView.	2. Creates a new points theme and corresponding point shapefile.
Alternative Courses: After step 2, the SDSS Operator clicks in ArcView on individual trees (see Section 7.1.4 for data accuracy levels) that are visible on an image backdrop (scanned 1 : 10 000 aerial photo or satellite image with resolution 2.5 m). Trees that are further away than ca. 30 m from a pipe can be neglected and should not necessarily be clicked on. The tree *size* attribute must then be classified on an ordinal scale using the numbers (1,2,3 and 4). After step 2, the SDSS Operator selects an image (scanned colour photo or satellite image). The system will then extract all the trees from the image and saves it in a vectorised polygon theme. The system will finally create point objects at the centre points of the polygons for representing the trees. The point tree object will have an area attribute that holds the area of the containing polygon. These (tree crown) area values can then be used directly in the tree size classification.	

Use case description: **Spatial join Trees & Pipes**

Actor Action	System Response
1. The SDSS Operator applies the spatial join operation in ArcView and specifies the two collections: *Trees* (main collection) and *Pipes* (associated collection).	2. Builds association: Tree object with its nearest pipe object and creates (and also populates) the *distance* and *userlinknr* fields in the *Trees* attribute table.
3. The SDSS Operator clicks the customised export button in ArcView to export the *Trees* attribute table. The SDSS Operator then drops the *distance* and *userlinknr* fields.	4. The *Trees* attribute table is exported in the query result collection file format (see Appendix A). *Distance* and *userlinknr* fields in the ArcView *Trees* attribute table are dropped.
5. The SDSS Operator clicks the *Import* navigation button in the Object Query Browser to import the new *Trees* collection.	6. The new updated *Trees* collection, which contains the *distance* and *userlinknr* attributes, is added to the Object Query Browser project.

FIGURE 6.13(a) Use case diagram descriptions: Pipe break susceptibility analysis – Trees (Part I)

Use case description: **Pipe break susceptibility analysis – for each tree**

Actor Action	System Response
1. The SDSS Analyst selects the *size* and *distance* attributes of the *Trees* collection as fuzzy controller input variables. He should specify also a fuzzy controller output variable, viz. *single_risk*. The fuzzy controller is then called.	2. The fuzzy controller program with its default tree project opens.
3. The SDSS Analyst can edit the fuzzy rules and membership functions, if necessary. He then activates the simulation.	4. The simulation starts and for each tree in the collection, a *single_risk* value is calculated.
5. When finished with the simulation, the SDSS Analyst clicks in the Fuzzy GUI of the Object Query Browser the *Update Active Collection* button.	6. The active query collection (i.e. the *Trees* collection) is updated with the fuzzy controller output data.

Use case description: **Pipe break susceptibility analysis – for each pipe**

Actor Action	System Response
2. The SDSS Analyst applies the navigation tool for building simple associations and specifies the *Pipes* collection as the main collection and *Trees* as the collection to be associated. The *userlinknr* attribute is specified for both association keys.	3. Builds a *one*-to-*many* association between *Pipes* and *Trees*.
4. The SDSS Analyst selects from Pseudo-Collection sub-GUI the *obj_Calc5* operation.	5. Calculates the accumulated pipe break susceptibility figure for each pipe taking into account the length of the pipe as will be discussed further in Section 6.2.3.2
Alternative Courses: After Step 5, the SDSS Analyst can refine the model result by querying all pipes with d ≤ 150 mm, so excluding the large diameter pipes that have a higher section modulus that may resist the bending moment caused by the tree roots. The SDSS Analyst can now also build a *one*-to-*many* association between the *Pipes* collection and the *Breaks_tree* collection (real pipe break occurrences caused by trees, see Section 7.1.3) and then apply the *collec_Calc2* operation to test and further calibrate the model. If necessary, the fuzzy rule base and membership functions should be adjusted to obtain a better correlation between model results and real pipe break occurrences.	

FIGURE 6.13(b) Use case diagram descriptions: Pipe break susceptibility analysis – Trees (Part II)

Use case description: **View pipe break susceptibility results and statistics**

Actor Action	System Response
1. The SDSS Analyst exports the pipe break susceptibility figures to the Graph subsystem and sorts them there in ascending order.	2. The cumulative pipe break susceptibility distribution graph is shown. This graph may also be of interest to the Public Works Administrator.
3. The SDSS Analyst can group the pipes into different *high* pipe break susceptibility classes and can then export these sub-collections.	4. The *high* pipe break susceptibility collection(s) are exported and are thus made persistent.
Alternative Courses: After Step 4, the SDSS Analyst can prepare thematic maps showing the pipes with *high* break susceptibility (as the result of trees). The Publics Works Administrator can then view these maps to assist him in the decision-making process regarding maintenance planning.	

FIGURE 6.13(c) Use case diagram descriptions: Pipe break susceptibility analysis – Trees (Part III)

6.2.3.2 Design

The operation *obj_Calc5* is a special operation that can be applied to calculate the accumulated pipe break susceptibility for a pipe as the result of nearby trees. The activity diagram for operation *obj_Calc5* is shown in Figure 6.14, describing the internal flow of the operation. The operation is implemented in the system with the following signature:

TreeOperations::obj_Calc5 (treedatarow:Treebranchdatarow, propnames:Collection, datarowcollec:Treebranchdatacollec, cpropnames:Collection)

As in all *obj_Calc* operations, the iteration through the collection is handled externally by the *calcObjrows* operations (see Section 5.4.3.1). The *calcObjrows* operation iterates through the *Pipes* collection and for each pipe in the collection the current pipe information (*treedatarow:Treebranchdatarow*) and the associated tree information (*datarowcollec:Treebranchdatacollec*), if any, are passed on as parameters to the *obj_Calc5* operation via the *calcTreebranchrow* operation (see Figure 5.28). The *obj_Calc5* operation then performs its calculations (see Figure 6.14) on the data received. The sum of the pipe break susceptibility figures from the associated tree information is determined. This sum figure, however, gives a too high value (see Section 6.2.3.1) and should be adjusted by applying the formula 6.1 to obtain the final accumulated pipe break susceptibility figure for the pipe. The accumulated pipe break susceptibility attribute of the pipe object in the *Pipes* collection is then updated accordingly.

The following empirically derived formula is used to determine the accumulated pipe break susceptibility for a pipe (as the result of nearby trees):

$$R = \sqrt{\frac{lo}{l}} \cdot Ro \qquad (6.1)$$

Where,

R = accumulated pipe break susceptibility

l_o = length of typical short pipe segment

l = pipe length

R_o = sum of the associated pipe break susceptibility figures

The formula uses the standard *sum* function (see Section 5.3.8) to calculate the sum (R_o) of all the associated pipe break susceptibility figures from the associated *Trees* collection and then multiplies this sum figure with a special factor that takes the pipe length into account. The factor implemented in formula 6.1 compares the length of each pipe with that of a typical short pipe segment of 50 m and makes the adjustment using the square root function. The square root function was chosen, since it gives a good result that lies somewhere in between the two extremes (see Table 6.1) of using the *sum* function only (which gives a too high figure) and the division of standard pipe segment by pipe length (which gives a too conservative figure).

TABLE 6.1 Example: Accumulating pipe break susceptibility figures (as the result of trees)

<table>
<tr><td colspan="2"><b>Case #1:</b> One tree near a pipe that has a typical short length (l_o).
The accumulated pipe break susceptibility figure (R) is in this case simply R_o.</td></tr>
<tr><td colspan="2"><b>Case #2:</b> Three trees of same size as in case #1 and also the same distance away from the pipe. The pipe is four times longer than the pipe in case #1, viz. $4l_o$. The pipe will have the following accumulated pipe break susceptibility figures:</td></tr>
<tr><td>$R = 3R_o$</td><td><i>sum</i> function → too high value</td></tr>
<tr><td>$R = \sqrt{\frac{l_o}{l}} \cdot R_o = \sqrt{\frac{l_o}{4 \cdot l_o}} \cdot (3R_o) = 1.5R_o$</td><td>$R = \sqrt{\frac{l_o}{l}} \cdot R_o$ → realistic middle value</td></tr>
<tr><td>$R = \frac{l_o}{l} \cdot R_o = \frac{l_o}{4l_o} \cdot (3R_o) = 0.75R_o$</td><td>$R = \frac{l_o}{l} \cdot R_o$ → too low value</td></tr>
</table>

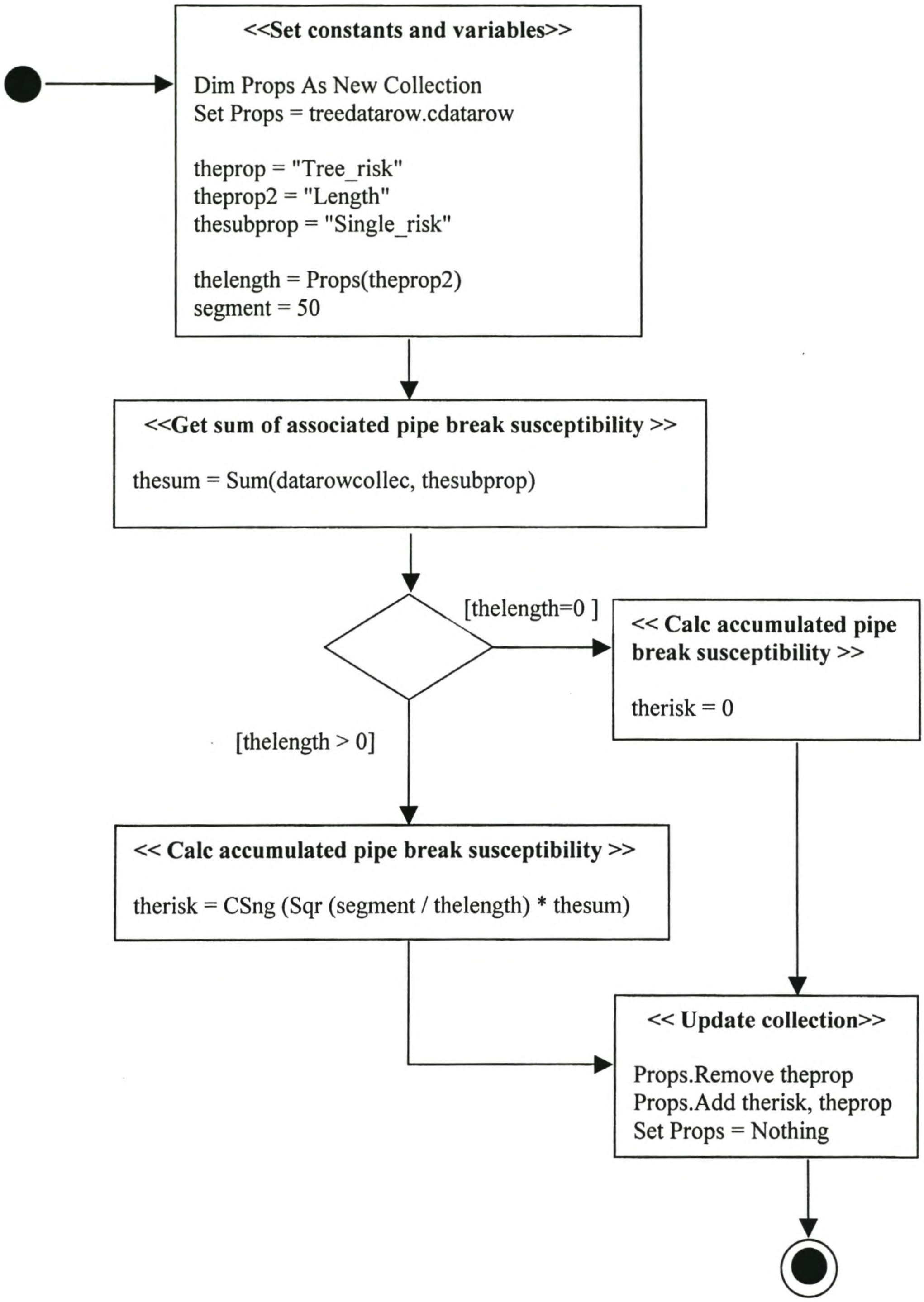

<<Set constants and variables>>
Dim Props As New Collection
Set Props = treedatarow.cdatarow
theprop = "Tree_risk"
theprop2 = "Length"
thesubprop = "Single_risk"
thelength = Props(theprop2)
segment = 50
<<Get sum of associated pipe break susceptibility >>
thesum = Sum(datarowcollec, thesubprop)
[thelength=0]
<< Calc accumulated pipe break susceptibility >>
therisk = 0
[thelength > 0]
<< Calc accumulated pipe break susceptibility >>
therisk = CSng (Sqr (segment / thelength) * thesum)
<< Update collection>>
Props.Remove theprop
Props.Add therisk, theprop
Set Props = Nothing

FIGURE 6.14 Activity diagram for operation *obj_Calc5*

Activity description: **<<Set constants and variables>>**

Creates a new temporary collection *Props* and sets it equal to the collection *treedatarow.cdatarow* which holds the current pipe information.

The attribute name of the attribute that must hold the accumulated pipe break susceptibility figure in the main collection is specified and stored in variable *theprop*. The attribute name of the attribute that holds the pipe length is specified and stored into variable *theprop2*. The pipe length can now be read from the *Props* collection and stored into variable *thelength*. The attribute name of the attribute (in the associated *Trees* collection) that holds the pipe break susceptibly figure as the result of a single tree is specified and stored into the variable *thesubprop*.

Lastly, the length of a typical short pipe segment in the network is specified and stored into variable *segment*.

Activity description: **<<Get sum of associated pipe break susceptibility>>**

The *sum* function (see Section 5.3.8) is applied to determine the sum of associated pipe break susceptibility figures in the associated *Trees* Collection.

Activity description: **<<Calc accumulated pipe break susceptibility>>**

The accumulated pipe break susceptibility figure is calculated taking the pipe length into account according to the formula 6.1. If the pipe length is zero as in the case of pumps and pressure reduction valves (which are modelled in the SDSS as pipe elements with zero pipe length) the accumulated pipe break susceptibility figure is set to zero, since otherwise a division by zero error will occur.

Activity description: **<<Update collection>>**

The current *treedatarow* of the *Pipes* collection is then updated with the newly calculated accumulated pipe break susceptibility figure (as the result of trees). Existing data, if any, are first removed. The temporary *Props* collection is then also removed from memory.

FIGURE 6.15 Activity diagram descriptions for operation *obj_Calc5*

6.3 REQUIREMENTS ANALYSIS AND DESIGN OF SUBSYSTEMS FOR PIPE BREAK IMPACT ASSESSMENT

In this section the detailed requirements analysis and design of the subsystems of the SDSS will be given for assessing pipe break impact with regard to water loss and property damage (see Section 1.1). It would eventually also be possible to model the inconvenience caused by a pipe break (see Section 1.1), provided that the necessary digital data can be captured (such as shut-off valve positions and traffic flow data on the roads). This section, however, will focus on the assessment of the two main impacts, viz. water loss and property damage, for which there were enough digital data available to construct accurate models. The pipe break impact assessment regarding water loss and property damage will be discussed separately under the sections that follow.

6.3.1 Water loss assessment

Special functionality is added to the SDSS to determine the outflow rate from a pipe break. Herewith pipe sections can be identified where high outflow can be expected during a pipe break. The network will be analysed under static load conditions, i.e. at night, when maximum water loss will occur as the result of a pipe break. High water loss pipe sections should receive high priority preventative maintenance in order to avoid large amounts of water loss should a pipe break in these regions. The unaccounted-for water (UFW) figures of the network can hereby also be reduced, since pipe bursts are a main constituent of the UFW figure (see Section 1.1). The theory on pipe break outflow will be discussed below.

6.3.1.1 Hydraulic modelling of a pipe break

When a pipe breaks in a network (under static load conditions), water from the reservoir/tank will reach the break location via different flow paths and will discharge from both ends of the broken pipe. Since the pressure at the discharge opening is atmospheric (the small velocity head at the discharge opening can be neglected), the available pressure head is therefore used up by friction losses in each flow path. This fact - in conjunction with the flow continuity principle, viz. flow entering a node must be equal to the flow leaving the node - can be used to calculate the pipe break outflow in a network.

Pipe break outflow can be modelled by placing a node at the break location (see pseudo-node #3 in Figure 6.16). The pressure drops to zero metres (atmospheric pressure) at this node #3 during pipe break. The friction loss (hf) along both flow paths must be equal to the available head (H) at node #3. The following conditions must therefore hold:

$$H = hf_{1\text{-}2} + hf_{2\text{-}3a} \qquad (6.2)$$

And,

$$H = hf_{1\text{-}2} + hf_{2\text{-}3b} \qquad (6.3)$$

Furthermore, according to the flow continuity equation,

$$Q_1 = Q_2 + Q_3 \qquad (6.4)$$

And,

$$Q_{out} = Q_2 + Q_3 \qquad (6.5)$$

The above equations cannot be solved explicitly, but can be solved by initial guessing of Q_2 and Q_3, and calculating Q_1 and the friction loss components $hf_{1\text{-}2,}$ $hf_{2\text{-}3a}$ and $hf_{2\text{-}3b.}$ The correct Q_2 and Q_3 can then be found by further iterations until equations 6.2 and 6.3 hold.

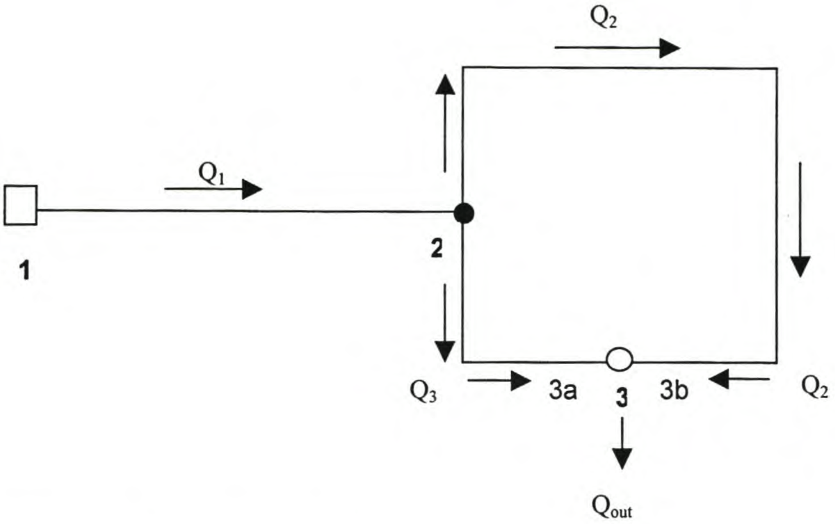


FIGURE 6.16 Modelling of pipe break outflow in a simple network

The pipe break outflow model assumes that there are no friction losses at the discharge opening. The actual outflow rate, especially in the beginning of a pipe break, may therefore be less than the outflow rate calculated by the model, because of the surrounding bedding material that offers resistance to the flow and must first be penetrated. After a while, when a large enough opening has been formed in both pipe and bedding material, the model results should then closely reflect the real situation.

The pipe break outflow in a large network can, however, not be determined so easily as demonstrated in the previous simple network, since the flow distribution in the network is then more complex. The only way to solve the problem is to use a water distribution system analysis program; this will be discussed in the following sections.

6.3.1.2 Numerical methods for balancing pressures and flows in a pipe network using a water distribution system analysis program

Solving for pressures and flows in a looped water distribution system is a mathematically non-linear problem for which a solution can only be found by an iteration technique. A method familiar to most engineers was introduced in 1936 by Cross (1936), but - except for a very small network - this single step method is not suitable due to its inefficiency in terms of computer time. A number of alternative algorithms using simultaneous iteration have since been developed. The algorithms are either based on the node method or on the loop method and differ in the way iteration results are updated after each trial solution. An evaluation of the different approaches is given by Ludewig (1989).

The three basic equations used in the balancing process are:

(a) The continuity equation (at each node inflow must be equal to outflow)

(b) The energy equation (energy at node 1 equals energy at node 2 plus friction loss)

(c) The head loss equation (modern programs use the Darcy - Weisbach equation to calculate friction loss in the pipes)

In the node-based algorithm the non-linear head loss equation for each pipe is linearised by means of the Newton-Raphson method using initial estimated flow rates and then substituted into the continuity equations at each node. This results in a system of linear equations with as many equations as there are nodes in the system. After the first solution of the linear equations, new flows and pressures can be calculated, and used as the basis for the linearisation of the equations for the second iteration, etc., until the changes in flow and pressure from one iteration to the next become insignificant.

The loop-based algorithm is basically the same, except that the linearised head loss equation for each pipe is substituted into the energy equation for each loop.

As both, the node-based and the loop-based algorithms have advantages and disadvantages, techniques have also been developed which may be termed hybrid node-loop methods. The method used in the EPANET program (see Section 5.4.2.3) is such an approach, developed by Todini and Pilati (1987), who called it the "Gradient Method". The method also begins with an initial estimate of flows in each pipe. At each iteration applying resistance linearisation, new nodal heads and flows are found. The iteration process continues until the sum of absolute flow changes relative to the total flow in all pipes is smaller than some tolerance (e.g. 0,001).

6.3.1.3 Calculation of outflow rate from a pipe break using a water distribution system analysis program

Water distribution system analysis programs, based on the theory of Section 6.3.1.2, can calculate the pressure head at the nodes and flows in the pipes of a network for given nodal demands (withdrawals). The demand figures must be specified at each node and the pressure head will then be calculated at each node, as well as the flow in each pipe though the network. The outflow as the result of a pipe break could be simulated by placing an additional node (pseudo-node) at the break location (as described in Section 6.3.1.1) and specifying a large demand figure at that newly placed pseudo-node. Since the demand, or rather the outflow rate, at the pseudo-node is unknown at this stage, an initial guess should be made. By guessing an initial too large outflow rate, a negative pressure head will occur at the pipe break node. This is because the head loss is also a function of the flow rate in the pipes. A smaller outflow rate should then be tried and should be reduced each time until the pressure head at the pipe break node is equal to zero metres (atmospheric pressure). This can be a very time-consuming process, but fortunately there are certain water distribution system analysis programs available, such as EPANET (see Section 5.4.2.3) that can do this iteration automatically, i.e. to reduce the initial outflow rate stepwise until the negative pressure head at the pipe break node eventually becomes zero metres. At that final balanced state small negative pressure can, however, still be present at some surrounding nodes. This often happens during pipe breaks and can cause destructive vacuums which can cause further breaks in the nearby pipe sections.

6.3.1.4 Requirements analysis

The SDSS Operator (see Figure 6.17) can access the EPANET hydraulic functions in the Hydraulic Library subsystem (see Section 5.2.2) via the Pipe Break Outflow sub-GUI of the Spatial GUI (see Section 5.4.2.3). The hydraulic functions can be applied to calculate the pipe break outflow rate at each pipe in the network (see Section 7.1.2 for data accuracy levels). The process is fully automated and should only be activated. When the calculation is completed, the resulting outflow values are stored as object attributes for each pipe and can then be instantly accessed and queried by the SDSS Analyst. The SDSS Analyst can, however, also perform a hydraulic pipe break outflow analysis on all or only selected pipes, which may be the case when changes are made to the network or if there is no SDSS Operator available to perform the initial analysis on all pipes in the network.

The SDSS Analyst can finally prepare the statistic figures, graphs and thematic maps that are of interest to the Public Works Administrator. With the Graph subsystem (see Section 5.2.2), a graph can be drawn of the sorted pipe break outflow rates, showing the cumulative pipe break outflow distribution (see Section 5.4.5). This graph can then be examined visually by the SDSS Analyst to determine a suitable upper-class interval for pipes with *high* pipe break outflow rates (see Section 7.2). The cumulative distribution graph of the outflow rates should be drawn for the small diameter pipes (d ≤ 150 mm) only, and the cut-off for the *high* outflow rate that can be obtained from this graph can then be used in the subsequent classification of pipes in the network. The cut-off for the *high* outflow rate will otherwise be set too high, with the result that only the large diameter pipes (that usually also break very seldom) will fall into this category.

The water loss assessment subsystem described in this section, applies a pipe break outflow model that calculates the average outflow rate for a pipe (see Section 6.3.1.5). The subsystem can therefore only be applied to pipe networks – it will give inaccurate results when assessing the water loss from breaks in long reservoir feeder lines.

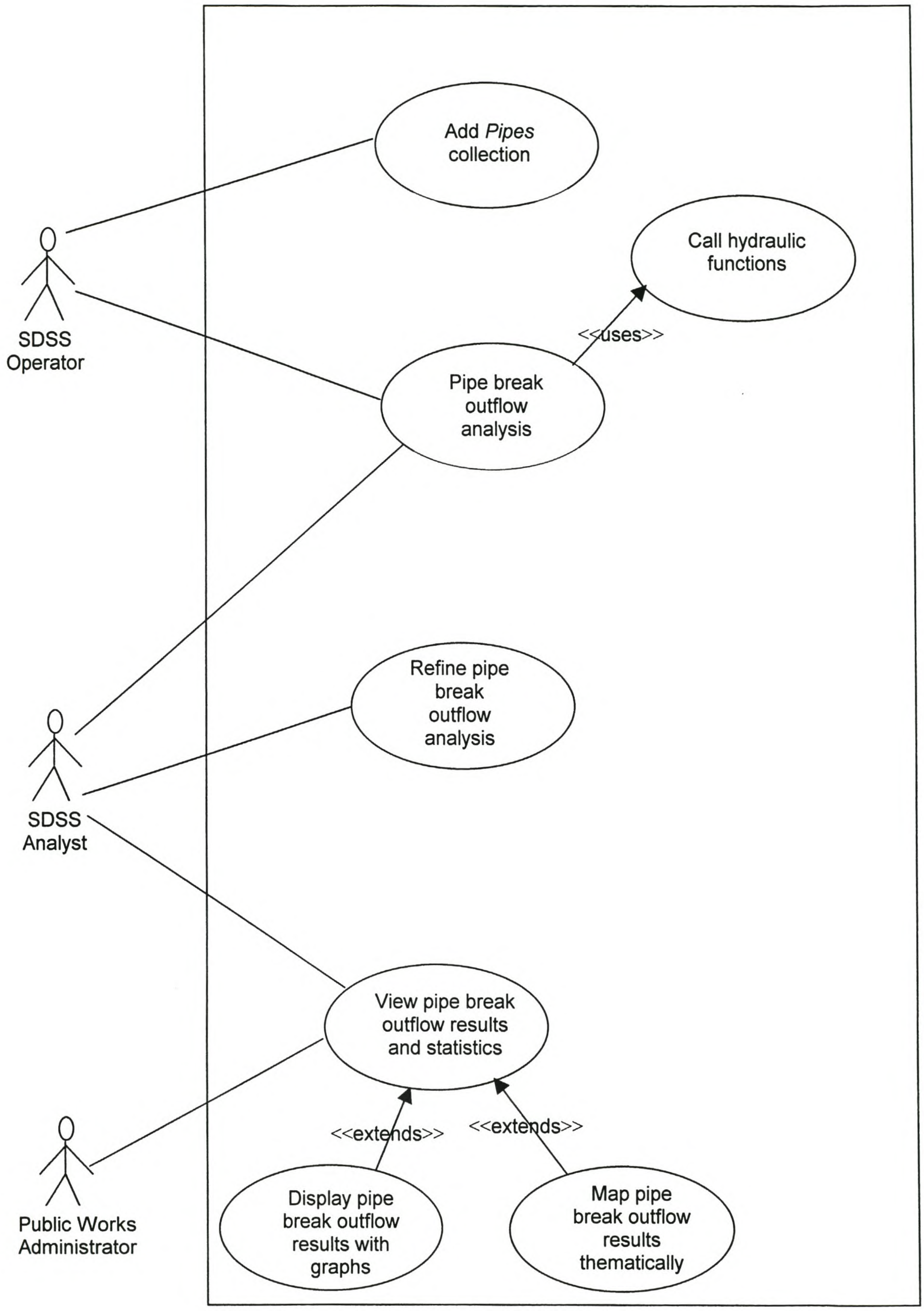

Add *Pipes* collection
Call hydraulic functions
<<uses>>
SDSS Operator
Pipe break outflow analysis
Refine pipe break outflow analysis
SDSS Analyst
View pipe break outflow results and statistics
<<extends>>
<<extends>>
Display pipe break outflow results with graphs
Map pipe break outflow results thematically
Public Works Administrator

FIGURE 6.17 Use case diagram: Pipe break impact assessment – Water loss

Use case description: **Add *Pipes* collection**

Actor Action	System Response
1. The SDSS Operator selects an existing *Pipes* collection to be loaded.	2. Adds the selected *Pipes* collection to the system.

Use case description: **Pipe break outflow analysis**

Actor Action	System Response
1. The SDSS Operator accesses the Pipe Break Outflow sub-GUI and specifies the start and end node attribute names. The attribute names to contain the outflow results are also specified. The hydraulic outflow analysis can then be started. The SDSS Analyst can also access the hydraulic functions to conduct the analysis on all or selected pipes only.	2. Calculates the outflow rate at the start and end node of each pipe. The average outflow rate for each pipe is then calculated. Finally, each pipe in the *Pipes* collection is then updated with the three outflow values that have been calculated.

Use case description: **Refine pipe break outflow analysis**

Actor Action	System Response
1. The SDSS Analyst can use the quick query facility to refine the pipe break outflow results by querying all pipes with d ≤ 150 mm.	2. The *Pipes* collection will be reduced accordingly.

Use case description: **View pipe break outflow results and statistics**

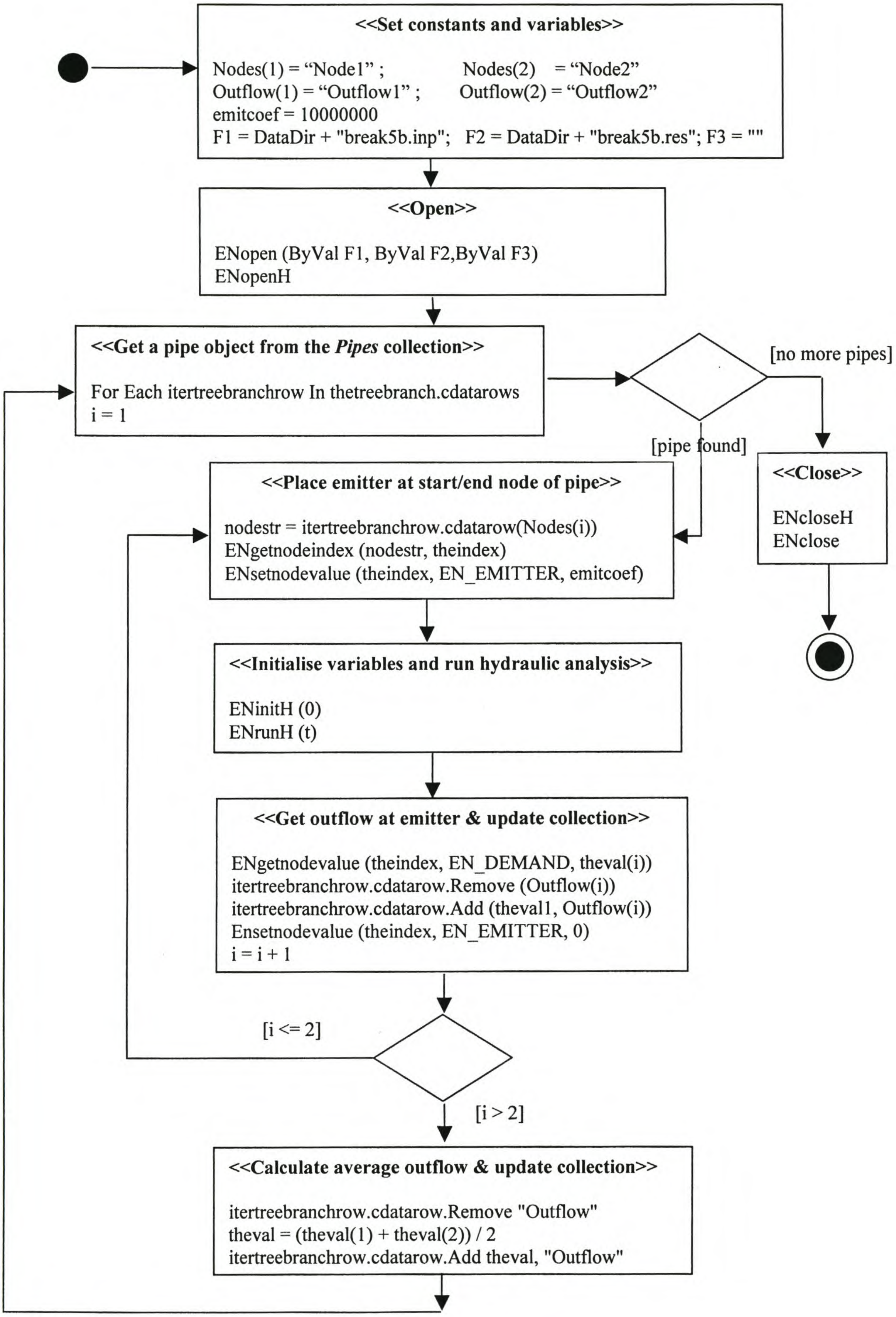
Actor Action	System Response
1. SDSS Analyst exports the pipe break outflow figures to the Graph subsystem and sorts them there in ascending order.	2. The cumulative pipe break outflow distribution graph is shown. This graph may also be of interest to the Public Works Administrator.
3. The SDSS Analyst can group the pipes with *high* pipe break outflow and can then export the sub-collection.	4. The *high* pipe break outflow collection is exported and is thus made persistent.
Alternative Courses: After Step 4, the SDSS Analyst can prepare thematic maps showing the pipes with *high* outflow rates. The Publics Works Administrator can then view these maps to assist him in the preventative maintenance planning.	

FIGURE 6.18 Use case diagram descriptions: Pipe break impact assessment – Water loss

6.3.1.5 Design

The hydraulic functions of the EPANET dynamic link library (see Sections 6.3.1.3 and 6.3.1.4) for calculating the pipe break outflows are called from the Pipe Break Outflow sub-GUI (see Section 5.4.2.3) via the *calcOutflow* operation. An activity diagram of the *calcOutflow* operation is shown in Figure 6.19, describing the internal flow of the operation. The operation is not part of the user accessible (and editable) functions and operations of the TreeOperations class (see Sections 5.2.2 and 5.4.3.1) to avoid accidental corruption. The iteration handling routines to iterate though the *Pipes* collection are located inside the operation (see Figure 6.19). The EPANET hydraulic functions that are called by the operation can automatically find the correct outflow rate that will give atmospheric pressure at the pipe break node, so that no manual iteration is necessary (see Section 6.3.1.3). The pipe break node (see Section 6.3.1.1) is modelled by these functions as an emitter. Emitters are devices associated with junctions that model the flow through a nozzle or orifice. In these situations the demand (i.e. the flow rate through the emitter) varies in proportion to the square root of the pressure at the junction. The constant of proportionality is termed the 'discharge coefficient'. For nozzles and sprinkler heads the manufacturer usually provides the value of the discharge coefficient in units of LPM per meter (flow through the device at a 1 m pressure drop). Emitters can be placed in the network to model flow through sprinkler systems, irrigation networks, fire hydrants and leakage in a pipe (US Environmental Protection Agency 1999).

For pipe break outflow analysis a very high emitter discharge coefficient (e.g. 1 000 000 LPM/m) should be specified. Hereby the EPANET program will neglect friction losses at the discharge opening, implying that a large enough opening has been formed in both pipe and bedding material. Breaks will be simulated at the start and end node of a pipe and an average outflow rate will be calculated for the pipe (see Figure 6.19). An extra pseudo-node can be placed at the pipe break location (see Section 6.3.1.1), but this can be very time consuming and will (in a pipe network) not give significantly more accurate results than calculating the average outflow rate as described.


<<Set constants and variables>>
<<Open>>
<<Get a pipe object from the *Pipes* collection>>
[no more pipes]
[pipe found]
<<Close>>
<<Place emitter at start/end node of pipe>>
<<Initialise variables and run hydraulic analysis>>
<<Get outflow at emitter & update collection>>
[i <= 2]
[i > 2]
<<Calculate average outflow & update collection>>

```
Nodes(1) = "Node1" ;          Nodes(2)   = "Node2"
Outflow(1) = "Outflow1" ;     Outflow(2) = "Outflow2"
emitcoef = 10000000
F1 = DataDir + "break5b.inp";  F2 = DataDir + "break5b.res"; F3 = ""
```

```
ENopen (ByVal F1, ByVal F2,ByVal F3)
ENopenH
```

```
For Each itertreebranchrow In thetreebranch.cdatarows
i = 1
```

```
ENcloseH
ENclose
```

```
nodestr = itertreebranchrow.cdatarow(Nodes(i))
ENgetnodeindex (nodestr, theindex)
ENsetnodevalue (theindex, EN_EMITTER, emitcoef)
```

```
ENinitH (0)
ENrunH (t)
```

```
ENgetnodevalue (theindex, EN_DEMAND, theval(i))
itertreebranchrow.cdatarow.Remove (Outflow(i))
itertreebranchrow.cdatarow.Add (theval1, Outflow(i))
Ensetnodevalue (theindex, EN_EMITTER, 0)
i = i + 1
```

```
itertreebranchrow.cdatarow.Remove "Outflow"
theval = (theval(1) + theval(2)) / 2
itertreebranchrow.cdatarow.Add theval, "Outflow"
```

FIGURE 6.19 Activity diagram for operation *calcOutflow*

Activity description: **<<Set constants and variables>>**

A very high emitter coefficient is specified, implying that there are no friction losses at the pipe break discharge opening. Attribute names of the start and end node of a pipe, output attribute names, and EPANET input and output file names are specified.

Activity description: **<<Open>>**

ENopen opens the EPANET Toolkit system to analyse a distribution system.
ENopenH opens the hydraulics analysis system.

Activity description: **<<Get a pipe object from the *Pipes* collection>>**

The *for each* loop iterates through the *Pipes* collection and exits the loop if no more pipes are found. The *Pipes* collection is represented by the *Treebranch.cdatarows*. The counter variable *i* is initialised and will be used as an index to access the attribute names in the *Nodes* array.

Activity description: **<<Place emitter at start/end node of pipe>>**

ENgetnodeindex retrieves the index of the node with *nodestr* as id label.
ENsetnodevalue sets the emitter coefficient at the corresponding node.

Activity description: **<<Initialise variables and run hydraulic analysis>>**

ENinitH initialises the hydraulic analysis settings.
ENrunH runs a single period hydraulic analysis.

Activity description: **<<Get outflow at emitter and update collection>>**

ENgetnodevalue gets the outflow at the emitter node.
The start/end node outflow figure of the pipe in the *Pipes* collection is then updated. Existing outflow data, if any, are first removed. The emitter node is then removed by resetting its emitter coefficient to zero.

Activity description: **<<Calculate average outflow & update collection>>**

The average of the start and end node outflows is calculated. The average outflow attribute of the pipe in the *Pipes* collection is then updated accordingly. An existing average outflow value, if present, is first removed.

Activity description: **<<Close>>**

ENcloseH closes the hydraulic analysis system, freeing all allocated memory.
ENclose closes down the EPANET Toolkit system.

FIGURE 6.20 Activity diagram descriptions for operation *calcOutflow*

6.3.2 Assessment of potential damage caused to property

The discharging water from a pipe break can cause extensive damage to nearby properties that are situated at unsuitable locations relative to the pipe, i.e. in the flow path of the streaming water.

Stands that are on a lower level than the road surface, such as those found at many hillside residential areas, are prone to water damage, especially if the pipe is situated under the sidewalk that is nearest to the house (see Figure 6.21).

If the pipe is under the sidewalk across the street, the impact of water damage will be far less, since the water must then first traverse the street and can only reach the property via the driveway. In this case, however, damage can still be caused if the stand is situated much lower than the road surface and the driveway slopes steeply downwards towards the garage and house (see Figure 6.22).

Damage caused to properties situated further downstream of a pipe break occurs very seldom, since the discharging water that flows in the side channel alongside the kerb of the street will be absorbed by the stormwater intakes. The pipe break impact assessment subsystem of the SDSS will therefore not include functionality for flood channelling analysis. The functionality of the subsystem will be centred around damage assessment of properties on stands situated at a lower level than the road surface but in close vicinity of the pipe break (see Figures 6.21 and 6.22).

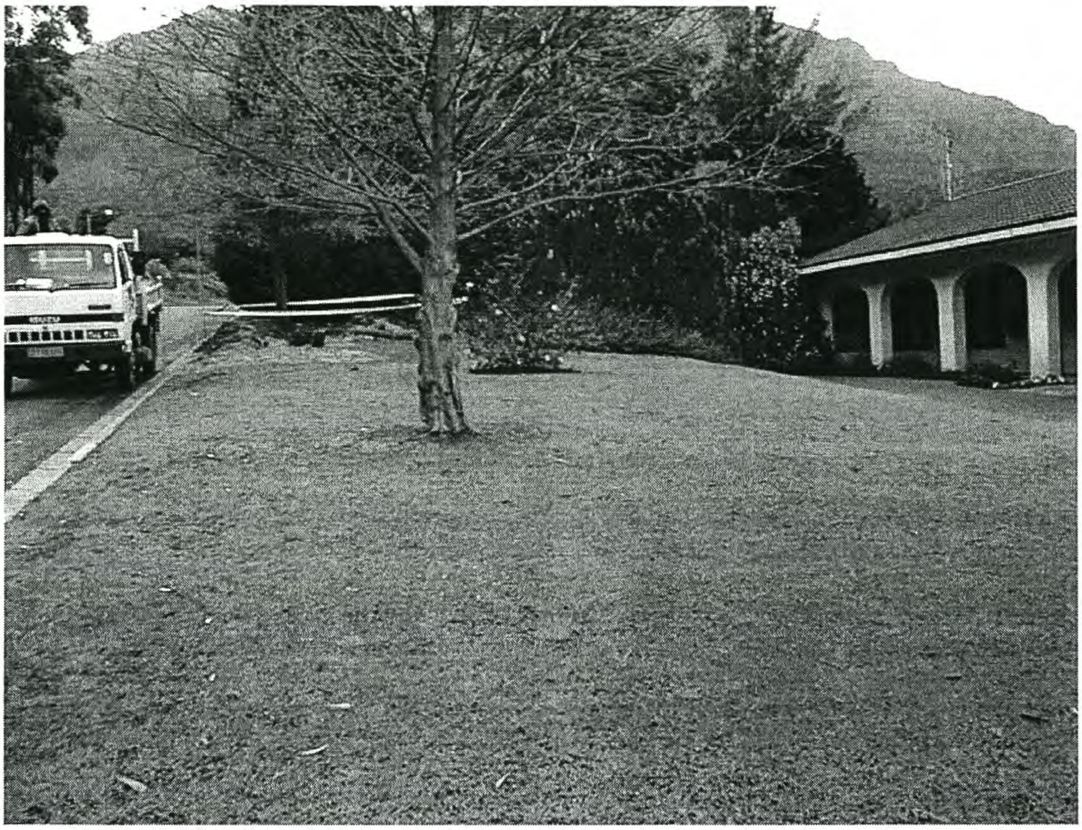


FIGURE 6.21 Stand with moderate cross-slope and pipe break at nearest sidewalk

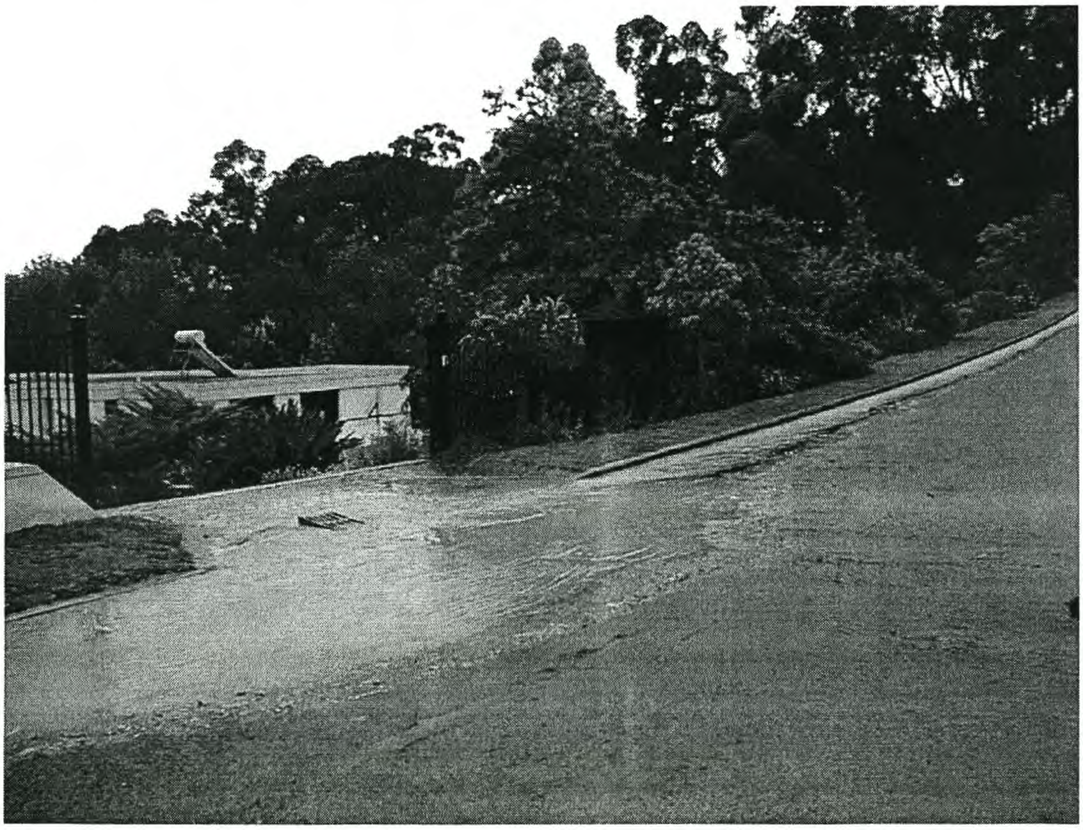


FIGURE 6.22 Stand with very steep cross-slope and pipe break at sidewalk across the street

6.3.2.1 Requirements analysis

The distance from the edge of a stand to the nearest pipe, as well as the cross-slope of the stand, are the two important factors that must be taken into account when analysing the pipe break impact on property. The data objects and notation used to model pipe break impact on property, taking into account the two main factors viz., *distance* away from pipe and *cross-slope* of stand, are depicted in Figure 6.23 (see Sections 7.1.1 and 7.1.2 for data accuracy levels). The cross-slope of a stand is the slope from the road surface down to the centre point of the stand. The greater the downhill cross-slope, the greater is the possibility that water will reach the property and cause damage (see Figures 6.21 and 6.22). The percentage cross-slope of a stand can be calculated with the following formula:

$$\mathrm{Stand}_{\mathrm{slope}} = \frac{Z_{\mathrm{standno}} - Z_{\mathrm{streetno}}}{d_{\mathrm{stand}}} \bullet 100 \qquad (6.6)$$

where,

$\mathrm{Stand}_{\mathrm{slope}}$ = the percentage cross-slope of the stand

Z_{standno} = the elevation at the centre of the stand indicated by the *standno* object (which is a short line object that has a stand number attribute)

Z_{streetno} = the elevation on the edge of the stand at the *streetno* object (which is a short line object that has a street number attribute)

d_{stand} = distance between the *standno* and *streetno* objects

The objects of the *Streetnos* collection can also be used as the starting point for the distance measurement d_{Pipe} from the edge of the stand to the nearest pipe (see Figure 6.23). The closer the pipe is to the edge of the stand, the greater will be the potential water damage during a pipe break.

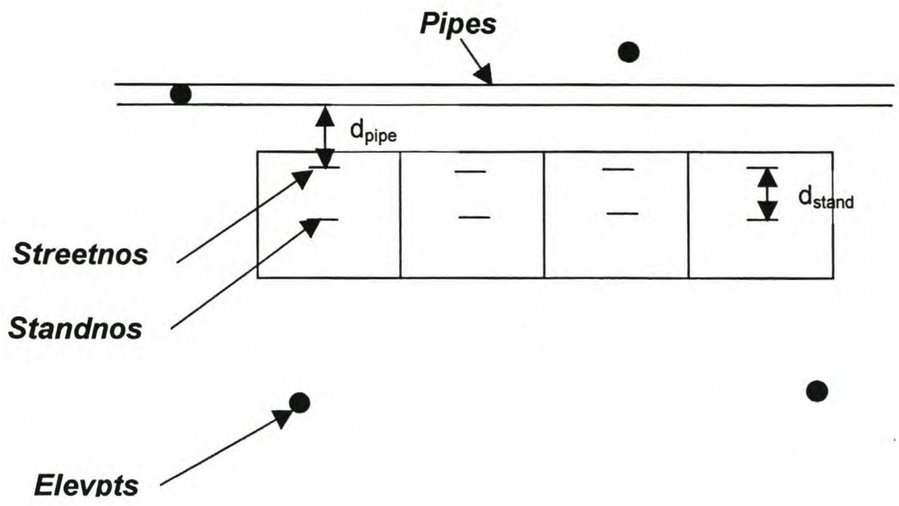


FIGURE 6.23 Description of data objects and notation used for pipe break impact assessment – Property damage

The SDSS Operator (see Figure 6.24) can access the interpolation subsystem of the SDSS via the Topographical Interpolation sub-GUI of the Spatial GUI (see Section 5.4.2.2) to interpolate the elevations for the objects in the *Streetnos* and *Standnos* collections. The objects of the *Elevpts* collection (see Figure 6.23) represent the points of known elevation that should be used in the interpolation process (see Section 5.4.2.2). A large enough rectangular extent (1000m x 1000m) should be specified for the interpolation search, so that the system can find enough known elevation points for interpolation and not have to extrapolate. A too large rectangular extent should, however, also not be specified, since this will slow down the interpolation process.

The SDSS Operator conducts a spatial join between *Streetnos* and *Standnos* in ArcView (see Figure 6.24). The spatial join operation finds for each *Streetno* object the nearest *Standno* object. The calculated distance to the nearest stand, viz. d_{stand} (see Figure 6.23), and the corresponding stand identifier are then automatically stored in newly created temporary fields in the ArcView *Streetnos* attribute table.

The SDSS Analyst can now analyse the pipe break impact on each stand (see Figure 6.24). The SDSS Analyst performs a spatial join operation in ArcView between *Streetnos* and *Pipes* to find the pipe nearest to the stand and the corresponding distance d_{pipe} as illustrated in Figure 6.23. Again, temporary fields for *userlinknr* and *distance* are automatically created and updated in the ArcView *Streetnos* attribute table. The SDSS Analyst can then import the newly updated *Streetnos* attribute table (that contains the results from both spatial join operations) into the Object Query Browser as described earlier in Section 5.4.2.1 (and should drop the temporary fields that have been created in the ArcView *Streetnos* attribute table, to avoid data duplication).

The SDSS Analyst can select the *obj_Calc6* operation from the Pseudo-Collection sub-GUI to calculate the cross-slope of each stand (viz. $Stand_{slope}$). The operation *obj_Calc6* contains the encoded formula 6.6.

The pipe break impact on a nearby stand is a fuzzy function of two input fuzzy variables, viz. *Stand_slope* (i.e. $Stand_{slope}$) and *Distance* (i.e. d_{pipe}) to the pipe. The fuzzy controller which is linked to the SDSS (see Section 5.2.2) can now be called via the Fuzzy GUI (see Section 5.4.4), where a standard project that has been set up especially for pipe break impact assessment can be opened. The input membership functions for *Stand_slope* and *Distance*; the output membership function *Stand_risk;* and also the rule base are shown in Section 7.4.2, when the SDSS will be applied to the study area of this dissertation. When the fuzzy controller has finished its fuzzy inference calculations, the *Streetnos* collection must be updated (see Section 5.4.4) so that each street number object in the *Streetnos* collection has its corresponding fuzzy controller output value assigned to it. This value represents the pipe break impact on the stand (at that street number).

The SDSS Analyst can now calculate the maximum pipe break impact on the stands along a pipe, taking into account the pipe break impact on all stands associated with the pipe. A *one*-to-*many* association (see Section 3.1.4.3) must be established between *Pipes* and its associated *Streetnos.* The link between the objects can be established by using the navigation tool for building simple associations (see Section 5.3.2) and specifying the *userlinknr* attribute for both association keys. The *obj_Calc4* operation, which has access to the *max* statistical function (see Section 5.3.8), can then be applied to find the maximum for each pipe of all its associated pipe break impact figures and then to store the figure in the *Max_stand_risk* attribute of the pipe object. The length of the pipe does not have such a great influence on the calculations as was the case with the pipe break susceptibility analysis due to trees (see Section 6.2.3). The reason for this is that the cross-slope of stands alongside the length of a pipe does not change so radically. If changes do occur, the *max* function, which is used in the impact assessment, will still pick up the stands with high break impacts, whereas the *average* function will balance out the result and will give a too low final pipe break impact figure.

The SDSS Analyst can finally prepare the statistic figures, graphs and thematic maps that are of interest to the Public Works Administrator. The Graph subsystem (see Section 5.2.2) can be used to plot the sorted pipe break impact figures, showing the cumulative pipe break impact distribution (see Section 5.4.5). This graph can then be examined visually by the SDSS Analyst to determine a suitable upper-class interval (see Section 7.2) for pipes with *high* pipe break impact (with regard to water damage caused to nearby properties).

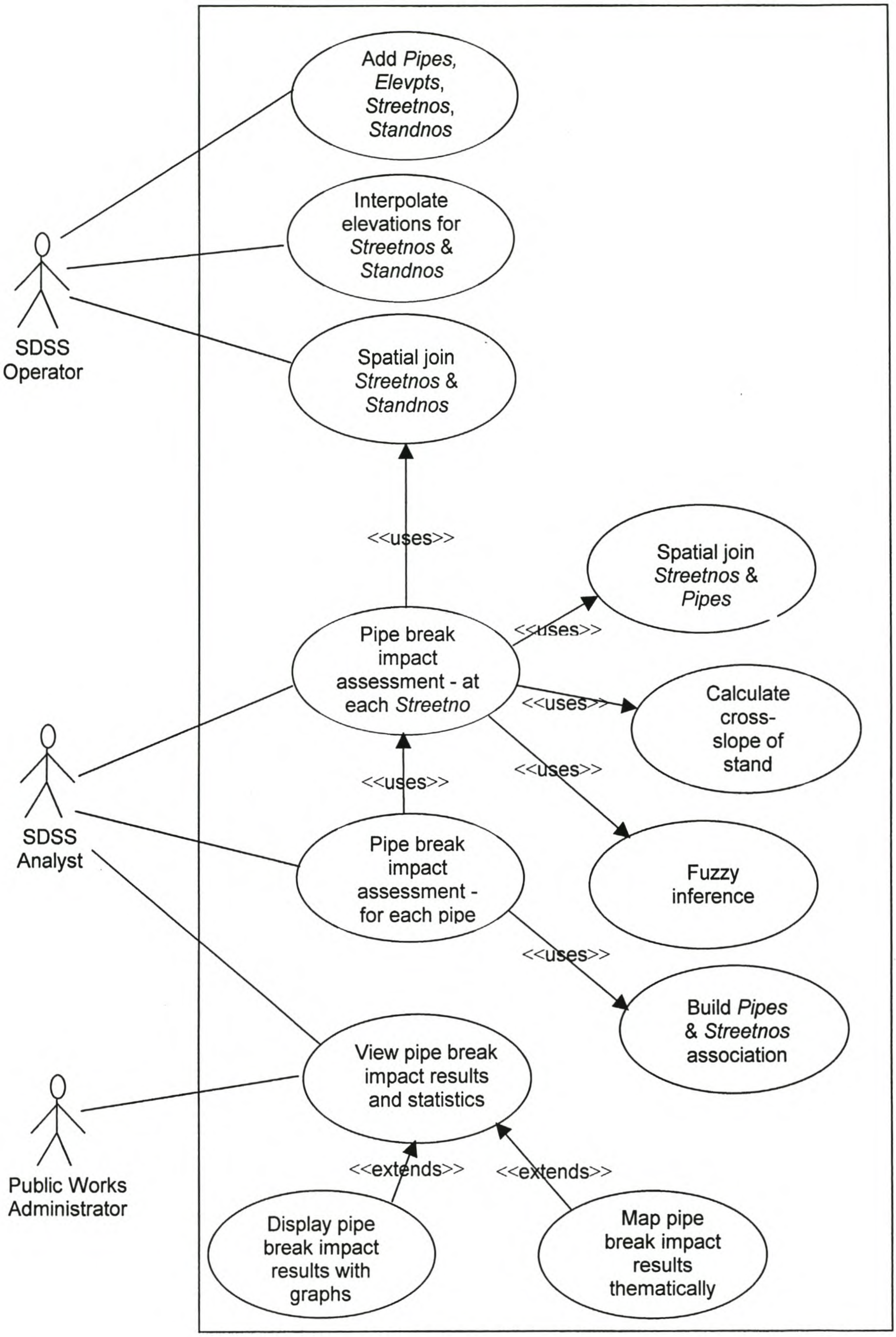

Add *Pipes*, *Elevpts*, *Streetnos*, *Standnos*
Interpolate elevations for *Streetnos* & *Standnos*
Spatial join *Streetnos* & *Standnos*
SDSS Operator
<<uses>>
Spatial join *Streetnos* & *Pipes*
<<uses>>
Pipe break impact assessment - at each *Streetno*
<<uses>>
Calculate cross-slope of stand
<<uses>>
<<uses>>
SDSS Analyst
Fuzzy inference
Pipe break impact assessment - for each pipe
<<uses>>
Build *Pipes* & *Streetnos* association
View pipe break impact results and statistics
Public Works Administrator
<<extends>>
<<extends>>
Display pipe break impact results with graphs
Map pipe break impact results thematically

FIGURE 6.24 Use case diagram: Pipe break impact assessment - Property damage

Use case description: **Add *Pipes, Elevpts, Streetnos and Standnos***

Actor Action	System Response
1. SDSS Operator selects an existing *Pipes* collection to be loaded in.	2. Adds the selected *Pipes* collection to the system.
3. SDSS Operator selects an existing *Elevpts* collection to be loaded in.	4. Adds the selected *Elevpts* collection to the system.
5. SDSS Operator selects an existing *Streetnos* collection to be loaded in.	6. Adds the selected *Streetnos* collection to the system.
7. SDSS Operator selects an existing *Standnos* collection to be loaded in.	8. Adds the selected *Standnos* collection to the system.

Use case description: **Interpolate elevations for *Streetnos and Standnos***

Actor Action	System Response
1. SDSS Operator accesses the Topographical Interpolation sub-GUI and the following must be specified: (a) The attribute names of the attributes in the *Streetnos* and *Standnos* collections that hold the x and y coordinate data; (b) Attribute names for the attributes to hold the interpolation results; (c) 1000 m x 1000 m rectangular extent should be specified for the interpolation search and (d) file containing the points of known elevation.	2. Interpolates elevation data for the objects contained in *Streetnos* and *Standnos.* The *Elevpts* point objects are the points of known elevation to be used in the interpolation process.

Use case description: **Spatial join Streetnos & Standnos**

1. The SDSS Operator applies the spatial join operation in ArcView and specifies the two collections: *Steetnos* (main collection) and *Standnos* (associated collection) to be joined spatially.	2. Builds association: *Streetno* object with its nearest *Standno* object and creates (and also populates) the *distance* and stand identifier fields in the *Streetnos* attribute table.

FIGURE 6.25(a) Use case diagram descriptions: Pipe break impact assessment – Property damage (Part I)

Use case description: **Pipe break impact assessment - at each *Streetno***

1. The SDSS Analyst applies the spatial join operation in ArcView and specifies the two collections: *Streetnos* (main collection) and *Pipes* (associated collection) to be joined spatially.	2. Builds association: Streetno object with its nearest pipe object and creates (and also populates) the *distance* and *userlinknr* fields in the *Streetnos* attribute table.
3. The SDSS Analyst clicks the customised export button in ArcView to export the *Steetnos* attribute table. The SDSS Analyst then drops the temporary fields.	4. The *Steetnos* attribute table is exported in the query result collection file format (see Appendix A). Temporary fields in the ArcView *Streetnos* attribute table are dropped.
5. The SDSS Analyst clicks the *Import* navigation button in the Object Query Browser to import the new *Steetnos* collection.	6. The updated *Steetnos* collection, containing the results from both spatial join operations, is added to the Object Query Browser project.
7. The SDSS Analyst applies the *obj_Calc6* operation to calculate the cross-slopes of the stands.	8. The cross-slopes (viz. *Stand_slope*) of the stands are calculated according to the formula 6.6.
9. The SDSS Analyst accesses the Fuzzy GUI and selects the *Stand_slope* and *Distance* attributes as fuzzy controller input variables. He should also specify the fuzzy controller output variable, viz. *Stand_risk*. The fuzzy controller is then called.	10. The fuzzy controller program opens.
11. The SDSS Analyst can edit the fuzzy rules and membership functions, if necessary. He then activates the simulation.	12. The simulation starts and for each object in the *Streetnos* collection, a *Stand_risk* value is calculated.
13. When finished with the simulation, the SDSS Analyst clicks in the Fuzzy GUI of the Object Query Browser the *Update Active Collection* button.	14. The active query collection (i.e. the *Streetnos* collection) is updated with the fuzzy controller output data.

FIGURE 6.25(b) Use case diagram descriptions: Pipe break impact assessment – Property damage (Part II)

Use case description: **Pipe break impact assessment - for each pipe**

Actor Action	System Response
1. The SDSS Analyst applies the navigation tool for building simple association and specifies the *Pipes* collection as the main collection and the *Streetnos* as the associated collection. The *userlinknr* attribute is specified for both association keys.	2. Builds a *one*-to-*many* association between *Pipes* and *Streetnos.*
3. The SDSS Analyst applies the *obj_Calc4* operation to find the maximum *Stand_risk* value of the associated *Streetnos.*	4. The maximum *Stand_risk* value of the associated *Streetnos* is found and the *Max_stand_risk* attribute of the object in the *Pipes* collection is updated with that value.

Use case description: **View pipe break impact results and statistics**

Actor Action	System Response
1. SDSS Analyst exports the pipe break impact figures to the Graph subsystem and sorts them there in ascending order.	2. The cumulative pipe break impact distribution of the network is shown in a graph. This graph may also be of interest to the Public Works Administrator.
3. The SDSS Analyst can group the pipes with *high* pipe break impact and can then export the sub-collection.	4. The *high* pipe break impact collection is exported and is thus made persistent.
Alternative Courses: After Step 4, the SDSS Analyst can prepare thematic maps showing the pipes with *high* break impact. The Publics Works Administrator can then view these maps to assist him in the decision-making process regarding maintenance planning.	

FIGURE 6.25(c) Use case diagram descriptions: Pipe break impact assessment – Property damage (Part III)

6.3.2.2 Design

Standard functions and operations of the Object Query Browser are applied as shown in the use case diagram of Figure 6.24. The design of the fuzzy rule base and membership functions, which is more a system application aspect, will be discussed in Section 7.4.2, when the SDSS will be applied to the study area of this dissertation.

6.4 REQUIREMENTS ANALYSIS AND DESIGN OF A SUBSYSTEM FOR COMBINED PIPE BREAK SUSCEPTIBILITY ANALYSIS AND IMPACT ASSESSMENT

A subsystem will be developed here for determining the combined pipe break susceptibility and impact on the network, i.e. taking into account both pipe break susceptibility and pipe break impact.

6.4.1 Requirements analysis

At this level the users of the system (i.e. the actors) would normally be the person in charge of the overall planning, viz. the Public Works Administrator and the SDSS Analyst who must prepare the summarised results.

The first step in combined pipe break susceptibility analysis and impact assessment will be to calculate the total pipe break susceptibility of the pipes (see Figure 6.26). The Multifactor Evaluation Process (MFEP), discussed in Section 5.3.7, will be used to calculate the total pipe break susceptibility of a pipe as the result of the three pipe break causes, viz. age, air pockets and trees. The recommended factor weights that should be assigned to the three pipe break causes are given in Table 7.2 of Section 7.5 (where the SDSS is applied to the study area of this dissertation).

A *Breaks_air_trees* collection of the study area containing actual pipe break occurrences as the result of air pockets and trees, if available, can be used to test and further refine the total pipe break susceptibility that is calculated by the model (see Section 5.3.9). If necessary, the factor weights can be adjusted to obtain a better correlation between model results and actual pipe break occurrences (see Section 7.2).

The second step in combined pipe break susceptibility analysis and impact assessment will be to calculate the total pipe break impact of the pipes (see Figure 6.26). The MFEP will be used to calculate a total pipe break impact figure consisting of water loss and impact on properties (see Table 7.3 for the recommended factor weights).

Several different combinations can then be analysed, which is the third step in combined pipe break susceptibility analysis and impact assessment. Some of these combinations are the following: total pipe break susceptibility and water loss; total pipe break susceptibility and property damage; or total pipe break susceptibility and total pipe break impact (see Figure 6.26). The final (combined) analysis result figure for a pipe is composed of the two individual analysis results figures combined. Percentages can be specified on how this composition should take place. A final results figure can typically be made up of 50% of the total pipe break susceptibility figure and 50% of the total pipe break impact figure.

The SDSS Analyst can finally prepare the statistic figures, graphs and thematic maps that are of interest to the Public Works Administrator. With the Graph subsystem graphs can be drawn of the sorted final analysis result figures, showing the cumulative distribution of the combined analysis. These graphs can then be examined visually by the SDSS Analyst to determine suitable upper-class intervals (see Section 7.2).

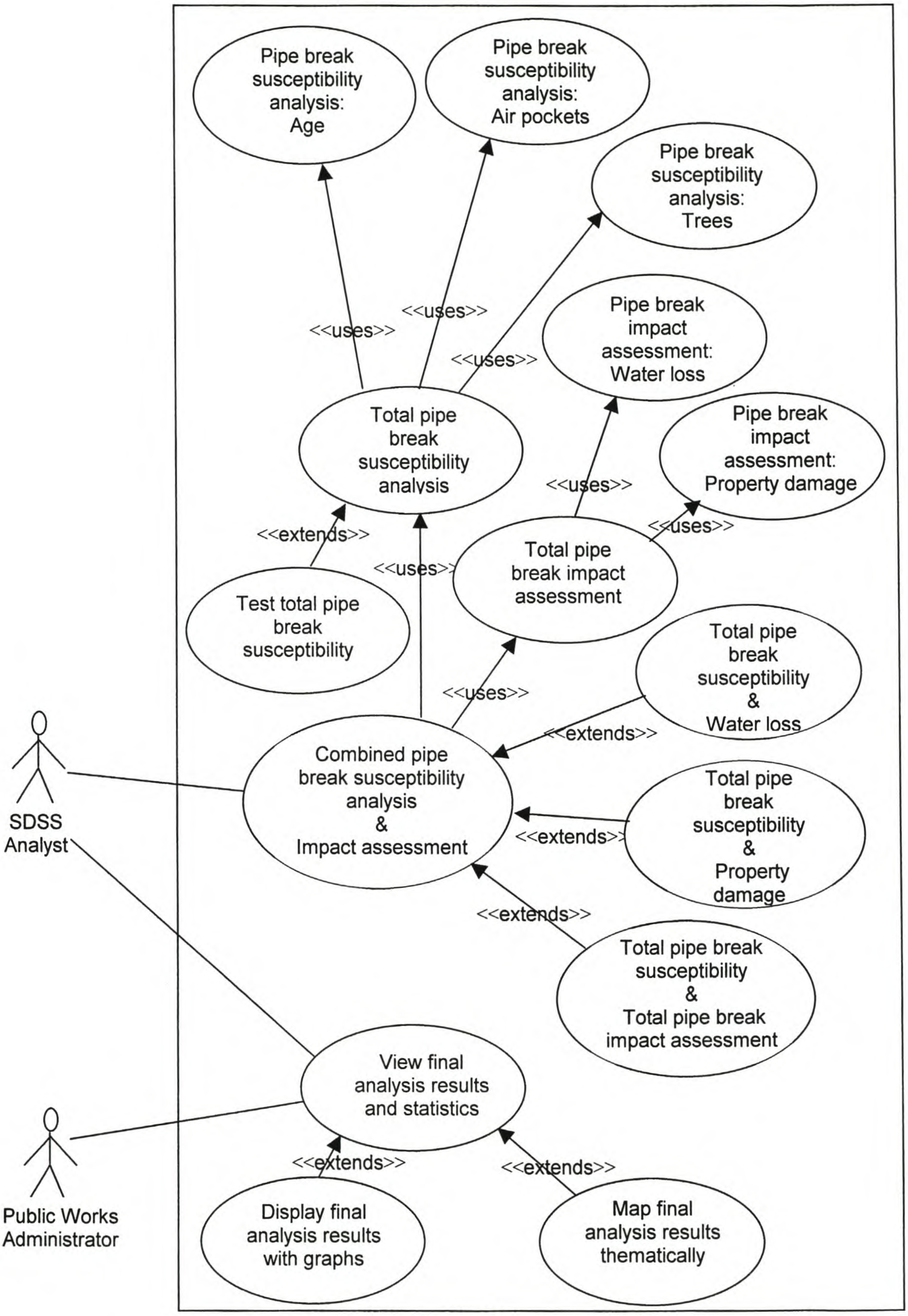

Pipe break susceptibility analysis: Age
Pipe break susceptibility analysis: Air pockets
Pipe break susceptibility analysis: Trees
Pipe break impact assessment: Water loss
Pipe break impact assessment: Property damage
<<uses>>
<<uses>>
<<uses>>
<<uses>>
<<uses>>
Total pipe break susceptibility analysis
<<extends>>
<<uses>>
Total pipe break impact assessment
Test total pipe break susceptibility
<<uses>>
Total pipe break susceptibility & Water loss
<<extends>>
SDSS Analyst
Combined pipe break susceptibility analysis & Impact assessment
Total pipe break susceptibility & Property damage
<<extends>>
<<extends>>
Total pipe break susceptibility & Total pipe break impact assessment
View final analysis results and statistics
Public Works Administrator
<<extends>>
<<extends>>
Display final analysis results with graphs
Map final analysis results thematically

FIGURE 6.26 Combined pipe break susceptibility analysis and impact assessment

Use case description: **Total pipe break susceptibility analysis**

Actor Action	System Response
1. The SDSS Analyst applies the *MFEP* operation and specifies *Pipes* as the target collection. The SDSS Analyst then specifies the output sub-collection containing the resultant pipes from the pipe break susceptibility analysis: Age. The factor weight (see Table 7.2) must also be specified.	2. For each pipe in *Pipes*, check if it is contained in the sub-collection. If so, then the specified factor weight is added to the *Total_break* attribute of the current pipe in the *Pipes* collection. The value of the *Total_break* attribute is initially zero.
3. The SDSS Analyst further applies the *MFEP* operation and specifies each of the output sub-collections containing the resultant pipes from the pipe break susceptibility analysis: Air pockets. The factor weights (see Table 7.2) must also be specified.	4. Continues with the accumulation process as described in step 2 for each specified sub-collection.
5. The SDSS Analyst further applies *MFEP* operation and specifies each of the output sub-collection containing the resultant pipes from the pipe break susceptibility analysis: Trees. The factor weights (see Table 7.2) must also be specified.	6. Continues with the accumulation process as described in step 2 for each specified sub-collection.
Alternative Courses: After step 6, the SDSS Analyst can now build a *one*-to-*many* association between the *Pipes* collection and the *Breaks_air_trees* collection (real pipe break occurrences as the result of air pockets and trees, see Section 7.1.3) to test and calibrate the system (see Section 5.3.9). If necessary, the factor weights should be adjusted to obtain a better correlation between model results and actual pipe break occurrences.	

FIGURE 6.27(a) Use case diagram descriptions: Combined pipe break susceptibility analysis and impact assessment (Part I)

Use case description: **Total pipe break impact assessment**

Actor Action	System Response
1. The SDSS Analyst applies the *MFEP* operation and specifies *Pipes* as the target collection. The SDSS Analyst then specifies the output sub-collection containing the resultant pipes from the pipe break impact assessment: Water loss. The factor weight (see Table 7.3) must also be specified.	2. For each pipe in *Pipes*, check if it is contained in the sub-collection. If so, then the specified factor weight is added to the *Total_impact* attribute of the current pipe in the *Pipes* collection. The value of the *Total_impact* attribute is initially zero.
3. The SDSS Analyst further applies the *MFEP* operation and specifies the output sub-collection containing the resultant pipes from the pipe break impact assessment: Property damage. The factor weight (see Table 7.3) must also be specified.	4. Continues with the accumulation process as described in step 2.

Use case description: **Combined pipe break susceptibility analysis & impact assessment**

Actor Action	System Response
1. The SDSS Analyst carries out a total pipe break susceptibility analysis as described in the use case: *Total pipe break susceptibility analysis.*	2. The total pipe break susceptibility analysis is carried out and the *Total_break* attribute for each pipe is updated by the system as described in the use case: *Total pipe break susceptibility analysis.*
3. The SDSS Analyst carries out a total pipe break impact assessment as described in the use case: *Total pipe break impact assessment.*	4. The total pipe break impact assessment is carried out and the *Total_impact* attribute for each pipe is updated by the system as described in the use case: *Total pipe break impact assessment.*
Alternative Courses: After step 4, the SDSS Analyst can apply the *obj_Calc2* operation to calculate a final (combined) analysis results figure which can consist of the following combinations: total pipe break susceptibility and water loss; total pipe break susceptibility and property damage; or total pipe break susceptibility and total pipe break impact. Percentages can also be specified for the composition of this final analysis results figure.	

FIGURE 6.27(b) Use case diagram descriptions: Combined pipe break susceptibility analysis and impact assessment (Part II)

Use case description: **View final analysis results and statistics**

Actor Action	System Response
1. The SDSS Analyst exports the final analysis results figures, obtained from the combined analysis, to the Graph subsystem and sorts them there in ascending order.	2. The cumulative distribution graph of the final analysis results is shown. This graph may also be of interest to the Public Works Administrator.
3. The SDSS Analyst can create different sub-collections of pipes that have *high* final analysis result figures and can then export them.	4. The pipe collection(s) are exported and are thus made persistent.
Alternative Courses: After Step 4, the SDSS Analyst can prepare thematic maps showing the pipes with *high* final analysis result figures. The Publics Works Administrator can then view these maps to assist him with the maintenance planning.	

FIGURE 6.27(c) Use case diagram descriptions: Combined pipe break susceptibility analysis and impact assessment (Part III)

6.4.2 Design

Standard functions and operations of the Object Query Browser are applied as shown in the use case diagram of Figure 6.26. The MFEP, which is used to summarise and combine the various analyses results, is implemented in the operation *tree_Calc4*. The design of this operation, including its activity diagram, is given in Section 5.3.7. An alternative approach would be to specify suitable fuzzy rules and use the fuzzy controller of the SDSS to combine the analyses results. At this advanced operating level, however, the complex analyses should all have been completed and the results contained in the independent pipe collections, can thus be summarised sufficiently with the simpler MFEP approach.

6.5 SUMMARY

In this chapter the subsystems of the SDSS have been designed for analysing pipe break susceptibility due to age, air pockets and tree roots, and for assessing pipe break impact with regard to water loss and property damage. A subsystem for combined pipe break susceptibility analysis and impact assessment has also been developed. The subsystem for analysing pipe break susceptibility due to age, estimates pipe age by the (very accessible and readily available) inauguration year attributes of the containing residential areas in which the pipes are located. The air pocket analysis subsystem analyses pipe break susceptibility as the result of seven possible types of air pocket formation that can occur in the network viz. (i) at long pipes that also have flat slopes; (ii) at pipes with blank ends; (iii) at pipes where there are changes in diameter; (iv) at nodal high points; (v) at nodal low points; (vi) at high points along the pipes; and (vii) at low points along the pipes. The subsystem for tree-root attack, applies fuzzy logic to model the effects of tree size and distance away from the pipe. The subsystem for assessing pipe break impact on nearby lower-lying properties also applies fuzzy logic in the modelling of topographical aspects such as the slope of a stand and distance away from the pipe break. The water loss assessment subsystem calls special hydraulic functions to determine the outflow rate from a pipe break. Finally, the combined pipe break susceptibility analysis and impact assessment subsystem provides functionality (such as the MFEP process), for combining the results from the various analyses conducted with the SDSS. In the next chapter the SDSS will be applied to a municipal water distribution system.

CHAPTER 7: APPLICATION OF THE SPATIAL DECISION SUPPORT SYSTEM TO A MUNICIPAL WATER DISTRIBUTION SYSTEM

In this chapter the SDSS is applied to the water distribution system of Paarl. An overview of the Paarl study area is given, focusing on the aspects relevant to the pipe break susceptibility analysis and impact assessment to be conducted. A pipe break susceptibility analysis is then conducted, taking into account the pipe break causes: age, air pockets and trees. A pipe break impact assessment with regard to water loss and property damage will then follow. Finally a combined pipe break susceptibility analysis and impact assessment will be conducted.

7.1 STUDY AREA – PAARL

The Paarl study area will now be discussed under the sections that follow. A soil survey of the area is given in Appendix C and can be used as a preliminary classification if the SDSS is to be extended to incorporate pipe break causes as the result of aggression and differential settlement (see Sections 4.3 and 4.6).

7.1.1. Age of residential areas and cadastral layout

The residential areas with their inauguration dates are shown in Figure 7.1. The data were obtained from the Paarl municipality. Residential area names with the endings *(Old)* and *(New)* distinguish the newer developments from the older parts within an area. The central area of Paarl has many old properties dating back to the 1800s. The year 1950 (an estimated average figure) is assigned to that area which is when, it is believed, the first properties with potable water connections were inaugurated. The thematic mapping functionality of the SDSS (see Section 5.4.6) was used to map the *old* (25 years and older) residential areas and the *new* residential areas (younger than 25 years), as depicted in Figure 7.1. This classification will be used in the pipe break susceptibility analysis (due to age, see Section 7.3.1). Accept

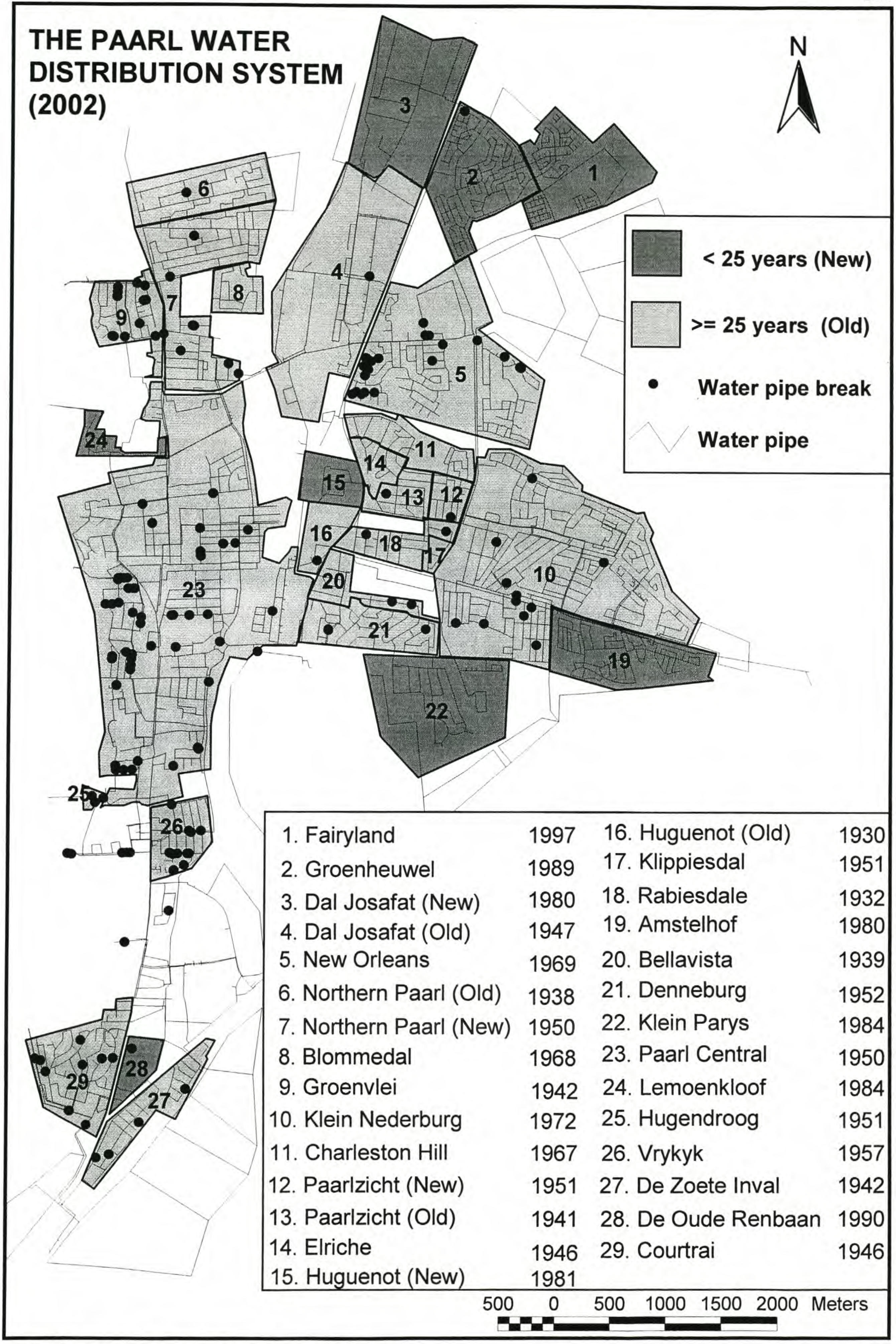


FIGURE 7.1 The Paarl water distribution system

for Paarl Central, the accuracy of the inauguration year data of each residential area is within one year - i.e. a time interval of one year is used in the date functions (see Section 6.2.1.2). This accuracy level is sufficient for defining the two classes that are used in the age classification, viz. ≥ 25 years (old) and < 25 years (new). The date functions used in the age analysis subsystem of the SDSS can easily be extended (see Section 6.2.1.2) to support smaller time intervals such as days, should more accurate digital data becomes available. Accurate digital data, such as the exact day, month and year of a pipe repair/replacement, will automatically start to become available when the SDSS is implemented, and the amount of digital age data will grow over the years. From a certain date on there will then be digital replacement dates recorded for each pipe in the network, with which a more detailed age classification can then be obtained.

Cadastral layout data consisting of ArcView shapefiles containing the outlines of each stand, have been obtained from the town planning division of the Paarl municipality. The data will be used in the property damage assessment (see Section 7.4.2). The accuracy of the obtained data is ± 0.01 to 0.1 m in the x and y directions, which is the usual accuracy required for town planning. Far less data accuracy would, however, also have been sufficient for the property damage assessment - the fuzzy overlaps of the membership functions have been set to 5% for the *stand slopes* and 2 m for the *distance* measurements (see Section 7.4.2) to make provision for uncertainty in the data classification. Only the outlines of the residential areas are shown in Figure 7.1 - the outlines of the individual stands, to be visible and distinguishable from one another, will require a larger scale plot. The cadastral layout data (see Figure 6.23) also includes street numbers (represented by a short line object at the centre and edge of a stand, with a street number attribute attached) and stand numbers (represented by a short line object at the centre of a stand, with a stand number attribute attached). The short line segments were generated and positioned automatically using ArcView.

7.1.2 Water distribution system layout

The Paarl and Wellington water distribution systems are semi-connected and fall under the same Drakenstein municipal administration. Water network data such as the topology, nodal elevations and peak-hour flows on the whole Paarl-Wellington water distribution system have been made available from the Paarl municipality for this research. The data consist of ArcView shapefiles and an EPANET input file of the network. The latter is used in the water loss assessment (see Section 7.4.1). This joint network consists of 4520 pipes. A pipe is defined as the connection between two nodes. Nodes are located at the junction points where pipes connect, where a pipe ends in a blank end, or also where a pipe diameter changes.

The other data, viz. residential age, pipe break occurrence data and elevation points data (also in ArcView shapefile format), are only available for the Paarl area (Mbekweni excluded). The pipe break susceptibility analyses and property damage assessment have therefore been restricted to the water distribution system of Paarl (Mbekweni excluded). This water distribution system consists of 3035 pipes and the layout is shown in Figure 7.1. The majority of the pipes in the network are asbestos-cement pipes.

The accuracy of the coordinates of the pipe nodes in the EPANET file for the hydraulic outflow analysis (to be conducted with the water loss assessment subsystem of the SDSS, see Section 7.4.1) is ± 0.1 to 0.5 m in the x, y and z directions. This is the usual accuracy that is required for balancing network pressures and flows in a water distribution system. The elevation points data (from which the z coordinates of the node data have been determined) also have an accuracy of ± 0.1 to 0.5 m in the x, y and z directions.

The pipe break susceptibility analysis subsystems of the SDSS and the property damage assessment subsystem also use pipe data and some of these subsystems also need elevation points data. These subsystems (especially the tree-break analysis subsystem, see Sections 7.1.4 and 7.3.3), however, do not require such high accuracy data as the water loss assessment subsystem. But, since these high accuracy data sets are required (and also readily available) for the water loss assessment subsystem, they can just as well be used also in the other subsystems.

7.1.3 Pipe break occurrences

A pipe break occurrence data file of the Paarl water distribution system has been made available by the Paarl municipality for this research. The file contains the pipe break occurrence data for the one-year period 1999/05/26 (when the municipality started with the digital data capturing of pipe break occurrences) to 2000/05/26. An extract of the file is shown in Figure 7.2, where only the relevant data columns that are applicable to this study are shown. The pipe break location is captured by the street address information (street name and street number). The additional information such as pipe diameter and other descriptive information such 'on sidewalk', 'in the street', etc. (which is encoded in the 'Position' field, see Figure 7.2) were helpful to identify the correct pipe if more than one pipe are located at the street address. Point objects representing the pipe breaks (see Figure 7.1) have then been created manually in ArcView, using the information described above to locate and capture the correct break location. This is a rather tedious data-capturing process and it is recommended that the municipality should in the future capture the break location using a GPS device. A GPS device with x, y accuracy of ± 2 to 3 m will be sufficient, since the breaks are stored and analysed (during the model calibration analyses) per pipe - the exact break location on the pipe is not that important for the model calibration analyses (only the pipe where the break occurred, must be captured). The ± 2 to 3 m accuracy level would be sufficient to match the break to the correct pipe even if more than one pipe are located near the break location (except when the two pipes are laid side by side underneath the same sidewalk with only a small gap in between, or when the break occurs close to a node connecting two or more pipes). Other important information contained in the pipe break occurrence file is the final break cause. The following break causes were recorded (and encoded in the 'Break_type' field): differential settlement, airburst, tree roots, damaged by construction activities and unknown causes.

There were 161 breaks in total during the one-year period, of which 112 were airbursts, 12 were caused by tree roots, 10 were caused by differential settlement and 27 breaks of which the break location could not be encoded, due to missing street numbers or too vague description of the break location. The majority of these 27 breaks were airbursts. Separate query collections have been created for each break cause containing the relevant break object data and the corresponding *userlinknr* attribute (i.e. the user link number) of the pipe where the break occurred.

Break_no	Date_of_ complaint	Time_of_ complaint	Street_name	Street_no	Inner_ diam	Outer_ diam	Pipe_ wall	Posi_ tion	Break_ type
599	19990528	12:00	Proteastraat	22	100	125	12	2	3
.	.	.	.	.	.	.	.	.	.

FIGURE 7.2 Extract from the pipe break data file

7.1.4 Trees

A total of 2427 trees were manually encoded (as described in Section 6.2.3) from ortho-photos of Paarl (scale 1:10 000, B&W) that were scanned in and used as backdrop images in ArcView.

The resolution of these ortho-photos (as well as the final scanned images) is approximately 2.5 m. With this resolution, large vehicles such as trucks and busses can clearly be identified on the images – the width of these vehicles (typically 2.7 m) is still measurable by zooming in and using the ArcView measuring tool. Cars can also be identified on the images, but their dimensions (especially the width) are too blurry to measure. Small trees with crown diameters 2.5 – 7 m (see Section 6.2.3.1) are clearly visible and measurable on the images. The 2.5 m resolution is sufficient for the tree-break analysis. The tree *size* intervals have been set to 7 m (see Section 6.2.3.1) and the fuzzy overlaps of the *distance* membership functions have been set to 5 m (see Section 7.3.3) to make provision for this data inaccuracy (of the source and possible data capturing errors), as well as for uncertainty in the data classification.

7.2 CLASSIFICATION OF SYSTEM RESULTS AND CALIBRATION OF MODEL

In this section classification methods will be discussed to classify the results obtained from the analyses and sub-analyses, conducted with the SDSS. The classification methods will be applied in Sections 7.3, 7.4 and 7.5 to classify the pipe break susceptibility analysis results and impact assessment results, as well as the results of the combined pipe break susceptibility analysis and impact assessment. Methods to test and further calibrate the pipe break susceptibility analysis model (i.e. the classification schema and decision rules used in the model), will also be discussed in this section. The same testing and calibration methods could also be applied to the pipe break impact assessment model, if data should become available on the extent of damage caused on surrounding lower-lying properties. The water loss assessment model, however, calls accurate hydraulic functions (see Section 6.3.1) – further calibration of this model would therefore not be necessary.

Cumulative distribution graphs can be very useful for data classification and can be plotted with the graphing functionality of the SDSS accessible via the Graphs GUI (see Section 5.4.5). The pipe break susceptibility factor at 80% of the total number of pipes (see Figure 7.3, the first cumulative distribution graph in this chapter) can be read off from the cumulative distribution graph. This factor can give a good first indication of pipes with *high* break susceptibility, since only 20% (i.e. 100% - 80% = 20%) of all pipes in the network have pipe break susceptibility factors ≥ the obtained cut-off factor. Cumulative distribution graphs will be drawn of the results of almost all the pipe break susceptibility analyses and impact assessments conducted in this chapter (except for the susceptibility to breakage due to age, where a more simpler classification schema will be sufficient, see Section 7.3.1).

When relatively few number of modelling factors are involved, such as in the tree-break analysis and property damage assessment (see Sections 7.3.3 and 7.4.2), then the classification cut-off factors obtained from the cumulative distribution graphs should be tested and possibly also further calibrated to ascertain that they correspond with the basic modelling concepts. For instance, for a 50-metre-long pipe section to be classified as having *high* break susceptibility as the result of nearby trees - there should at least be one tree in the vicinity that definitely poses a moderate risk (see Section 7.3.3 and Figure 7.6).

Actual pipe break occurrence data can be used to test and further calibrate the pipe break susceptibility models in the following way: The *high* pipe break susceptibility collection can be obtained by querying the *Pipes* collection for pipes with break susceptibility factor greater ≥ the 20% cut-off factor (that has been read off from the cumulative distribution graph, as described above). This *high* pipe break susceptibility collection can then be tested and refined (can also further be classified into a *very high* and an *extremely high* class) by associating the collection with the pipe break occurrence data collection and then apply the *collec_calc2* operation (see Section 5.3.9) to calculate and display the calibration statistics. The calibration statistics consisting of the factors for sharpness, success rate and reliability, should be evaluated collectively when deciding on the class intervals to use (see Section 5.3.9). Classification and calibration should at first be done for single pipe break causes, before attempting to classify and calibrate the models for multiple pipe break causes. In general, a total pipe break susceptibility analysis (i.e. analysing multiple pipe break causes) will give better calibration statistics than a single pipe break susceptibility analysis (i.e. the analysis of only one pipe break cause), since there is seldom only a single pipe break cause involved. The SDSS user can define his own criteria when evaluating the calibration factors. The recommended (there is no standard yet) calibration factors for the total pipe break susceptibility classes should typically be the following: *high* (sharpness factor: 45% and success rate factor: 80%), *very high* (sharpness factor: 25% and success rate factor: 60%) and *extremely high* (sharpness factor: 10%, success rate factor: 25% and reliability factor: 30%). Reliability factors need only be evaluated for the extremely high classes, and should then increase to almost 100% when the class interval cut-offs are further raised, confirming that the pipe break susceptibility analysis models function correctly (see Section 5.3.9). The success rate factors will decrease when the sharpness factors increase in sharpness (i.e. a reduction in the numerical sharpness factor values, see Section 5.3.9), since fewer pipes will then be identified by the system - obviously some breaks will then be missed. Part of the classification and model calibration is to find an optimum balance between the sharpness and success rate factors. If this balance (i.e. the recommended calibration factors as given above) cannot be established then the weighting factors, decision rules and fuzzy rules applied in the pipe break susceptibility analysis models, should be edited.

The combined pipe break susceptibility analysis and impact assessment that will be conducted in Section 7.5 and also one of the pipe break susceptibility analyses (see Section 7.3.2) require MFEP weighting factors (see Section 5.3.7 and Tables 7.1, 7.2 and 7.3) to be specified for the query results categories and sub-categories. These user-specified MFEP weighting factors will form part of the SDSS decisions rules, and may need to be further calibrated if a balance between the sharpness and success rate factors cannot be established. The combined pipe break susceptibility analysis and impact assessment consists of a total pipe break susceptibility analysis and a total pipe break impact assessment. MFEP weighting factors should be specified for both of these total analyses, which will then be accumulated (i.e. an accumulated weighted evaluation will be conducted, see Section 5.3.7) to obtain the total pipe break susceptibility factor and the total pipe break impact factor (see Section 7.5). The final results factor is then calculated which is typically made up of 50% of the total pipe break susceptibility factor and 50% of the total pipe break impact factor. When classifying these two total factors and also the final results factor into classes of *high*, *very high* and *extremely high*, then the composition of their underlying MFEP weighting factors in the various categories and sub-categories will play a significant role (see Section 7.5). For instance, for a pipe to be classified as having *high* total pipe break susceptibility - it should comply with all sub-categories of one of the main pipe break cause categories (viz. age, air pockets or trees) and must at least also fall into the first sub-category of another main pipe break cause category (see Table 7.2).

The *high*, *very high* and *extremely high* results classes defined in the analyses all have class intervals ≥ a certain cut-off value, i.e. the *high* class interval does not end at the beginning of the *very high* class interval, but continues right through (the *very high* class interval also continues right through). This type of classification thus differs somewhat from the standard practice, but is ideal to be used in the MFEP process. The MFEP process first checks if the pipe is a member of the *high* class, and if so the MFEP weighting factor that has been assigned to that class, will be added to the accumulated weighted evaluation value of the pipe (see Section 5.3.7). The pipe is then further checked if it is also a member of the other two classes and the accumulated weighted evaluation value of the pipe will be updated accordingly.

7.3 PIPE BREAK SUSCEPTIBILITY ANALYSIS

The results of the pipe break susceptibility analysis on the water distribution system of Paarl will be discussed in this section. The pipe break susceptibility analysis of potential breakage caused by age, air pockets and trees will be discussed separately under the sections that follow.

7.3.1 Susceptibility to breakage due to age

Of the 3035 pipes in the study area, 2684 pipes were located in residential areas and could be analysed by the pipe break susceptibility subsystem of the SDSS for age analysis (see Section 6.2.1). The age of the pipes located outside residential areas is unknown and could thus not be analysed by the SDSS. These pipes, however, are mostly water mains with large diameters (greater than 150mm) and would therefore break very seldom. The age analysis subsystem was then used in the way described by the use case diagram of Section 6.2.1.1 to identify (and export to a new sub-collection) the *old* pipes (25 years and older) in the network that are located in the *old* residential areas (see Figure 7.1). The age analysis subsystem has identified 2127 pipes that fall into the *old* pipes class. In general, these pipes are more susceptible to breaking than the younger pipes located in the newer residential areas (see Figure 7.1), because of the longer period during which they could have undergone deterioration (see Section 4.1). The sharpness factor (see Section 5.3.9), which is 70% (viz. 2127 pipes out of 3035), is not very impressive, as could be expected from an age analysis. The results from the analysis can therefore not be used on their own, but must be used in conjunction with the results from other pipe break susceptibility analyses (see Section 7.5). Age, however, has a significant influence on the pipe break occurrences in the study area, since there were only two break occurrences of the pipes in the newer residential areas (see Figure 7.1). This clearly confirms the assumption that newer pipes are less susceptible to breaking than older pipes because of the pipe material, which is still more intact and can thus resist stronger forces. The pipes in the newer areas will therefore almost all get a lower total pipe break susceptibility factor assigned, because they are not included in the *old* pipes class (the 2127 pipes that were selected earlier by the age analysis subsystem), which is an important category that makes up one third of the total pipe break susceptibility figure (see Section 7.5).

7.3.2 Susceptibility to breakage caused by air pockets

A pipe break susceptibility analysis with the focus on breaks caused by air pockets was conducted on the Paarl water distribution system by applying the pipe break susceptibility subsystem of the SDSS for air pocket analysis (see Section 6.2.2 and Section 7.1.2 for the system layout data used).

The subsystem was used to establish sub-collections of the pipes in the study area, according to the classification of the 7 main air-pocket formation types and the sub-types shown in Table 7.1. Smaller factor weights are assigned to the sub-collections of the low point types (both nodal low points and low points along the pipes), because low points do not have such a great influence on air-pocket formation than high elevation points and the other types shown in the Table 7.1. The sub-collections of pipes with changes in diameter (see Table 7.1) contain many duplicate pipes resulting from the expansion (see Section 5.3.4) of the *Nodes* collection, which has a *many*-to-*many* association with the *Pipes* collection (see Section 6.2.2.1). This is why some of the sub-collections of that type have such high numbers for pipes found. The duplicate pipes are, however, not removed from the sub-collections, because they should also be taken into account when calculating the break susceptibility for a pipe (as the result of air pockets).

The pipe break susceptibility (as the result of air pockets) is then calculated for each pipe by applying the MFEP (see Section 6.2.2.1) on the pipe sub-collections obtained from the classification shown in Table 7.1. The cumulative distribution of the pipe break susceptibility factors (resulting from the air pocket analysis) for the 3035 pipes in the Paarl network is shown in Figure 7.3. A pipe break susceptibility factor ≥ 0.17 could be classified as *high*, since only 20% of all pipes have pipe break susceptibility factors of that magnitude (see Figure 7.3). This classification for *high*, however, may be too sharp since many of the real airburst occurrences in the network during the test period will then be missed when calibrating the system (see Section 5.3.9). A pipe break susceptibility factor ≥ 0.1 would be a better classification for *high* pipe break susceptibility (as the results of air pockets). The SDSS found 1600 pipes that fall into this *high* pipe break susceptibility class out of the 3035 pipes in the Paarl network (giving a sharpness factor of 53%). Of the 112 airbursts that occurred during the test period (see Section 7.1.3), 84 where also located along pipes that fall in the *high* pipe break susceptibility class (giving a success rate factor of 75%). Two more classes can be defined, viz. a *very high* and an *extremely high* pipe break susceptibility, to narrow in the results, obtaining better sharpness factors. The cut-offs for the pipe break susceptibility factors for these two classes are ≥ 0.16 and ≥ 0.3 respectively. The SDSS identified 790 pipes out of the 3035 pipes in the network that fall into the *very high* class (giving a sharpness factor of 26% and a success rate factor of 51%). The *extremely high* class consists of only 91 pipes out of the 3035 pipes (which gives a very fine sharpness factor of 3% and a success rate factor of almost 20%). The reliability factor (see Section 5.3.9) is already 22% for the *extremely high* class and for pipe break susceptibility factors of between 0.4 and 0.45, it reaches 100%, which means that the pipe break susceptibility model (as the results of air pockets) functions correctly, since all the pipes identified by the SDSS as having these extremely high break susceptibility factors of over 0.4 have actually also burst.

TABLE 7.1(a) Air-pocket formation types and sub-types (Part I)

Air-pocket formation types and sub-types	**Number of pipes found**	**Factor weight**
TYPE 1: LONG PIPES WITH FLAT SLOPE Length ≥ 220; Diam ≤ 100; Velocity ≤ 0.7; -3 ≤ Slope ≤ 3	155	$\frac{1}{7}$
TYPE 2: PIPES WITH BLANK ENDS		
Type 2.1 All blank ends	394	$\frac{1}{7}\cdot\frac{1}{2}=\frac{1}{14}$
Type 2.2 Blank end pipes that slope downwards Diam ≤ 100; Velocity ≤ 0.6; -3 ≤ Slope ≤ 1.5	157	$\frac{1}{7}\cdot\frac{1}{2}=\frac{1}{14}$
TYPE 3: PIPES WITH CHANGES IN DIAMETER		
Type 3.1 All changes in diameter	3541	$\frac{1}{7}\cdot\frac{1}{5}=\frac{1}{35}$
Type 3.2 Medium changes in diameter (Max_diam – Min_diam) ≥ 50; Min_diam ≤ 100;	2082	$\frac{1}{7}\cdot\frac{1}{5}=\frac{1}{35}$
Type 3.3 Large changes in diameter (Max_diam – Min_diam) ≥ 75; Min_diam ≤ 100;	1172	$\frac{1}{7}\cdot\frac{1}{5}=\frac{1}{35}$
Type 3.4 Medium transition in diameter (Max_diam – Min_diam) ≥ 50; Min_diam ≤ 100;	58	$\frac{1}{7}\cdot\frac{1}{5}=\frac{1}{35}$
Type 3.5 Large transition in diameter (Max_diam – Min_diam) ≥ 75; Min_diam ≤ 100;	36	$\frac{1}{7}\cdot\frac{1}{5}=\frac{1}{35}$
TYPE 4: PIPES WITH NODAL HIGH POINTS		
Type 4.1 All nodal high points	456	$\frac{1}{7}\cdot\frac{1}{3}=\frac{1}{21}$
Type 4.2 Nodal high points & medium diameter Diam ≤ 150; Velocity ≤ 0.7;	298	$\frac{1}{7}\cdot\frac{1}{3}=\frac{1}{21}$
Type 4.3 Nodal high points & small diameter Diam ≤ 100; Velocity ≤ 0.7	229	$\frac{1}{7}\cdot\frac{1}{3}=\frac{1}{21}$
TYPE 5: PIPES WITH NODAL LOW POINTS		
Type 5.1 All nodal low points	466	$\frac{1}{7}\cdot\frac{1}{3}\cdot\frac{1}{3}=\frac{1}{63}$
Type 5.2 Nodal low points & medium diameter Diam ≤ 150; Velocity ≤ 0.7;	283	$\frac{1}{7}\cdot\frac{1}{3}\cdot\frac{1}{3}=\frac{1}{63}$
Type 5.3 Nodal low points & small diameter Diam ≤ 100; Velocity ≤ 0.7	223	$\frac{1}{7}\cdot\frac{1}{3}\cdot\frac{1}{3}=\frac{1}{63}$

Units: Diam [mm], Max_diam [mm], Min_diam [mm]
Length [m], Velocity [m/s], Slope [%]

TABLE 7.1(b) Air-pocket formation types and sub-types (Part II)

Air-pocket formation types and sub-types	Number of pipes found	Factor weight
TYPE 6: PIPES WITH HIGH POINTS		
Type 6.1 All high points	165	$\frac{1}{7}\cdot\frac{1}{3}=\frac{1}{21}$
Type 6.2 High points & medium diameter Diam ≤ 150; Velocity ≤ 0.7;	128	$\frac{1}{7}\cdot\frac{1}{3}=\frac{1}{21}$
Type 6.3 High points & small diameter Diam ≤ 100; Velocity ≤ 0.7	117	$\frac{1}{7}\cdot\frac{1}{3}=\frac{1}{21}$
TYPE 7: PIPES WITH LOW POINTS		
Type 7.1 All low points	131	$\frac{1}{7}\cdot\frac{1}{3}\cdot\frac{1}{3}=\frac{1}{63}$
Type 7.2 Low points & medium diameter Diam ≤ 150; Velocity ≤ 0.7;	94	$\frac{1}{7}\cdot\frac{1}{3}\cdot\frac{1}{3}=\frac{1}{63}$
Type 7.3 Low points & small diameter Diam ≤ 100; Velocity ≤ 0.7	71	$\frac{1}{7}\cdot\frac{1}{3}\cdot\frac{1}{3}=\frac{1}{63}$

Units: Diam [mm], Velocity [m/s]

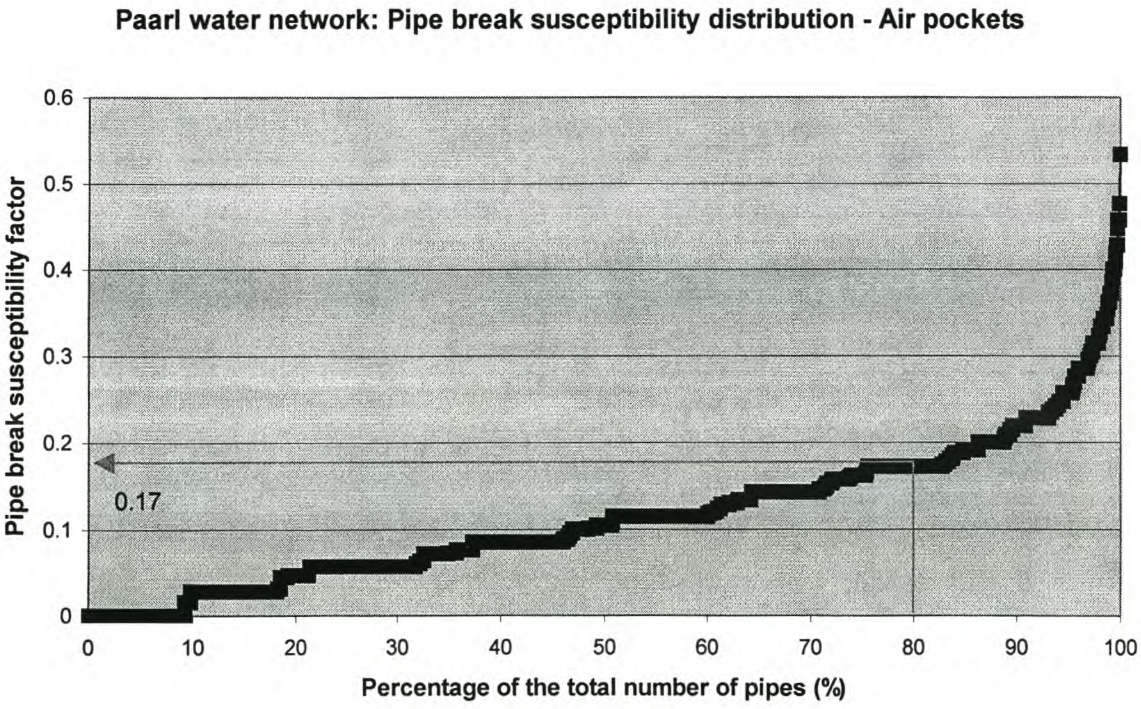


FIGURE 7.3 Pipe break susceptibility factor vs. percentage of total number of pipes

7.3.3 Susceptibility to breakage caused by trees

A pipe break susceptibility analysis with a special focus on breaks caused by trees was conducted on the Paarl water distribution system, using the pipe break susceptibility subsystem of the SDSS for tree break analysis (see Section 6.2.3).

The pipe-breaking potential of each captured tree (see Section 7.1.4) was determined using the fuzzy controller that is linked to the SDSS (see Section 5.4.4). The fuzzy controller input variables are tree *Size* and *Distance* from pipe. The fuzzy controller output variable is *Single_risk* and the following fuzzy rule base was used (see also Sections 2.2.2 and 2.2.3):

1.0 if Size is small and Distance is very_near then Single_risk is low
1.0 if Size is small and Distance is near then Single_risk is very_low
1.0 if Size is medium and Distance is very_near then Single_risk is moderate
1.0 if Size is medium and Distance is near then Single_risk is low
1.0 if Size is medium and Distance is relative_far then Single_risk is very_low
1.0 if Size is large and Distance is very_near then Single_risk is high
1.0 if Size is large and Distance is near then Single_risk is moderate
1.0 if Size is large and Distance is relative_far then Single_risk is low
1.0 if Size is large and Distance is far then Single_risk is very_low
1.0 if Size is very_large and Distance is very_near then Single_risk is very_high
1.0 if Size is very_large and Distance is near then Single_risk is high
1.0 if Size is very_large and Distance is relative_far then Single_risk is moderate
1.0 if Size is very_large and Distance is far then Single_risk is low
1.0 if Size is very_large and Distance is very_far then Single_risk is very_low

The two membership functions for the input variables viz. *Size* and *Distance*, that were used in the rule base are shown below in Figures 7.4 and 7.5.

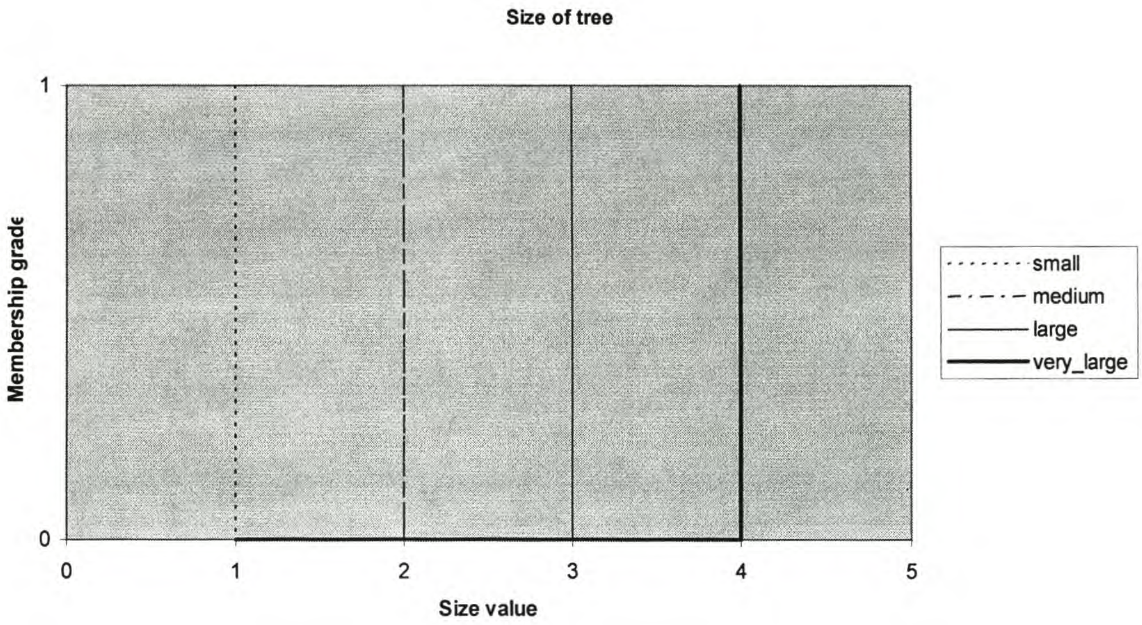


FIGURE 7.4 A singleton-type membership function for input variable *Size*

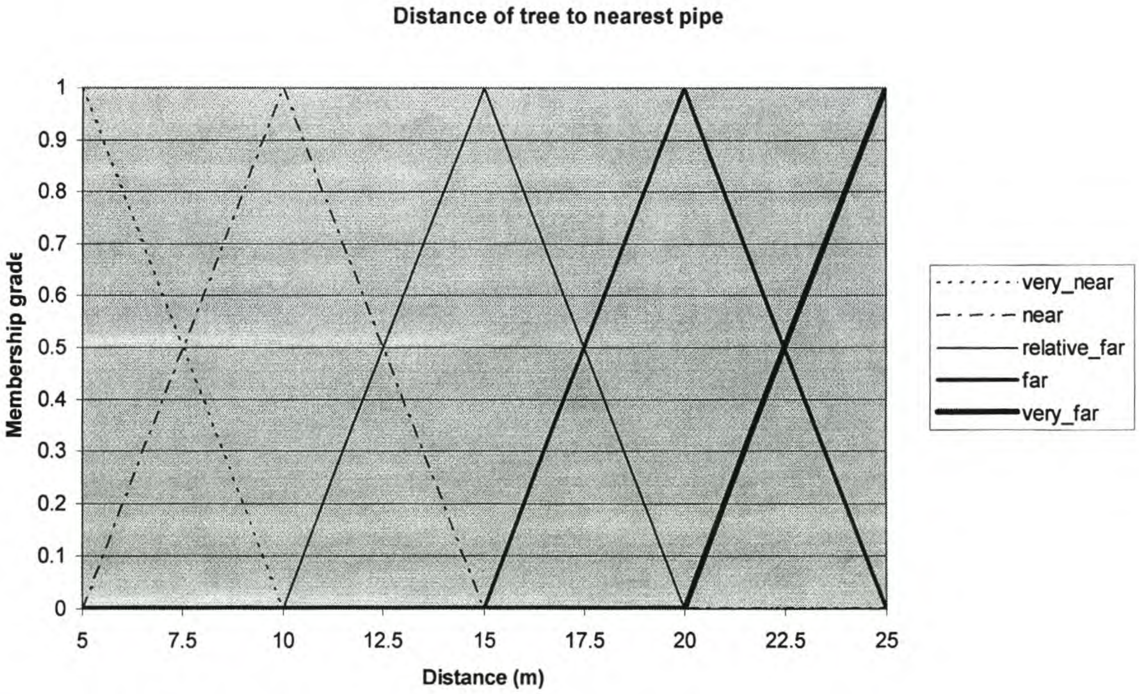


FIGURE 7.5 Membership function for input variable *Distance*

The membership function for the output variable *Single_risk*, representing the pipe break susceptibility as the result of a single tree, is shown below in Figure 7.6. The shape of the *Single_risk* output function is not so important, since the fuzzy controller uses the simplified centroid defuzzification method (see Section 2.2.3) and therefore only needs the centroidal x-coordinates at 0, 0.25, 0.5, 0.75 and 1. A singleton-type output member function could thus also have been specified, but the shape as shown in Figure 7.6 is more expressive.

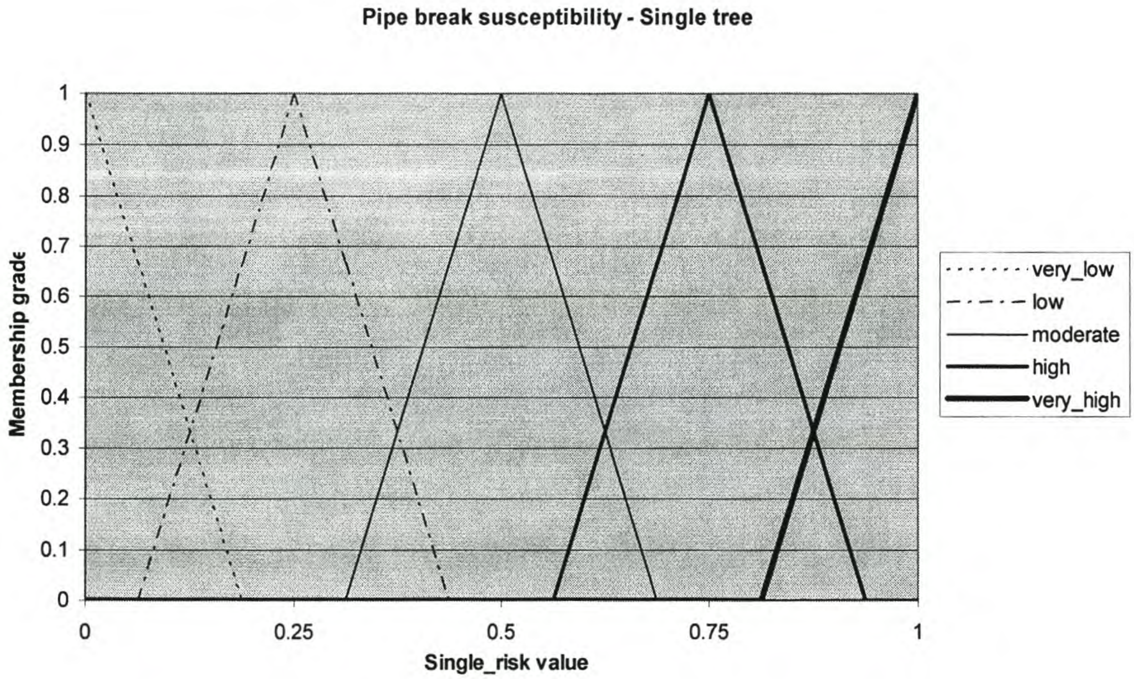


FIGURE 7.6 Membership function for output variable *Single_risk*

The output value *Single_risk,* which is a fuzzy function of both tree *size* and *distance,* can be displayed as a control surface as shown in Figure 7.7. As can be seen from the plot, the larger the tree size, the greater the *Single_risk* value becomes. The *Single_risk* value also increases when the distance to the nearest pipe decreases. It can also be seen from the plot that the larger the tree size, the further the tree can be away from nearest pipe and still have a significant pipe-breaking influence (i.e. significant *Single_risk* value).

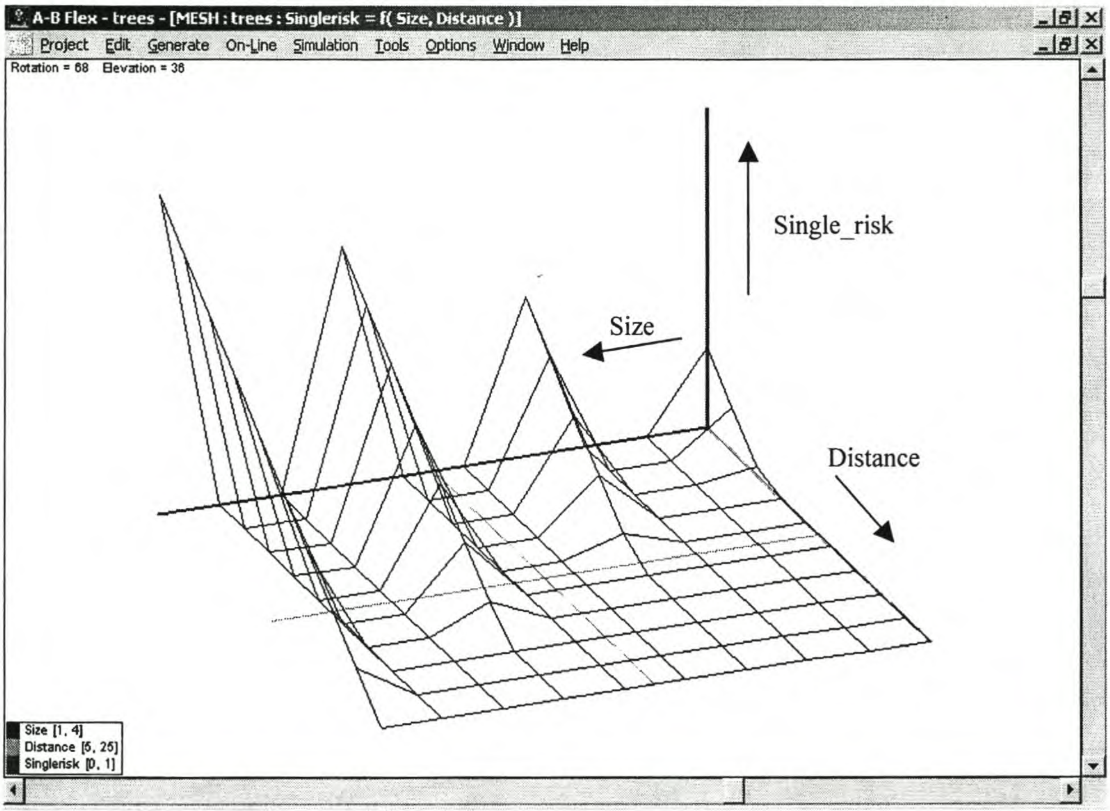


FIGURE 7.7 Control surface: *Single_risk* = f(*Size*, *Distance*)

The pipe break susceptibility factor (as the result of trees) for each pipe in the Paarl network was then calculated using the tree analysis subsystem (see Section 6.2.3.1). Since only the small diameter pipes are really susceptible to breaking, as discussed earlier in Sections 4.5, the pipe break susceptibility analysis (as the result of trees) will be restricted to small diameter pipes (d ≤ 150 mm). The large diameter water mains often cross through woody areas and will then only give misleadingly high pipe break susceptibility factors, since in most cases they should be able to resist the tree-root forces. The cumulative distribution of the pipe break susceptibility factors (with regard to trees) for the 2489 small diameter pipes in the Paarl water network is shown in Figure 7.8. A pipe break susceptibility factor ≥ 0.30 could be classified as *hig*h, since only 20% of all small diameter pipes have pipe break susceptibility factors of that magnitude (see Figure 7.8). This classification, however, may not be sharp enough, since it implies that a 50-metre-long pipe section with a tree located alongside it, which poses only a low to moderate risk (and more on the low side, see Figure 7.6), will then also be classified under the pipes with *high* pipe break susceptibility. A pipe break susceptibility factor ≥ 0.50 will be a better classification for pipes of *high* pipe break susceptibility, since alongside a 50-metre-long pipe section there should now at least be one tree that definitely poses a moderate risk (see Figure 7.6). The SDSS found 342 small diameter pipes that fall into this *high* pipe break susceptibility class; this makes up 14% of all small diameter pipes and 11% of all 3035 pipes (all diameters) in the Paarl water network (giving a sharpness factor of 11%). All 12 tree-related pipe breaks that occurred in the test period (see Section 7.1.3) were also located along pipes that fall into the *high* pipe break susceptibility class (giving an impressive success rate factor of 100%). Two more classes can be defined, viz. a *very high* and an *extremely high* pipe break susceptibility class. The cut-offs for the pipe break susceptibility factors for these two classes are ≥ 1.5 and ≥ 2.0 respectively. The SDSS identified 45 pipes out of the 3035 pipes in the network that fall into the *very high* class (giving a very fine sharpness factor of 1.5%). It was also found that 75% of all tree-related breaks were located along these pipes (i.e. a success rate factor of 75%). The *extremely high* class consists of only 15 pipes out of 3035 pipes (which gives a very fine sharpness factor of 0.5% and a success rate factor of almost 50%). The reliability factor (see Section 5.3.9) is already 33% in the *extremely high* group and at a pipe break susceptibility factor of over 2.5, it reaches 80% which means that the tree break susceptibility model of the SDSS functions correctly, since almost all the pipes identified by the SDSS as having these extremely high beak susceptibility factors of over 2.5 have actually also broken.

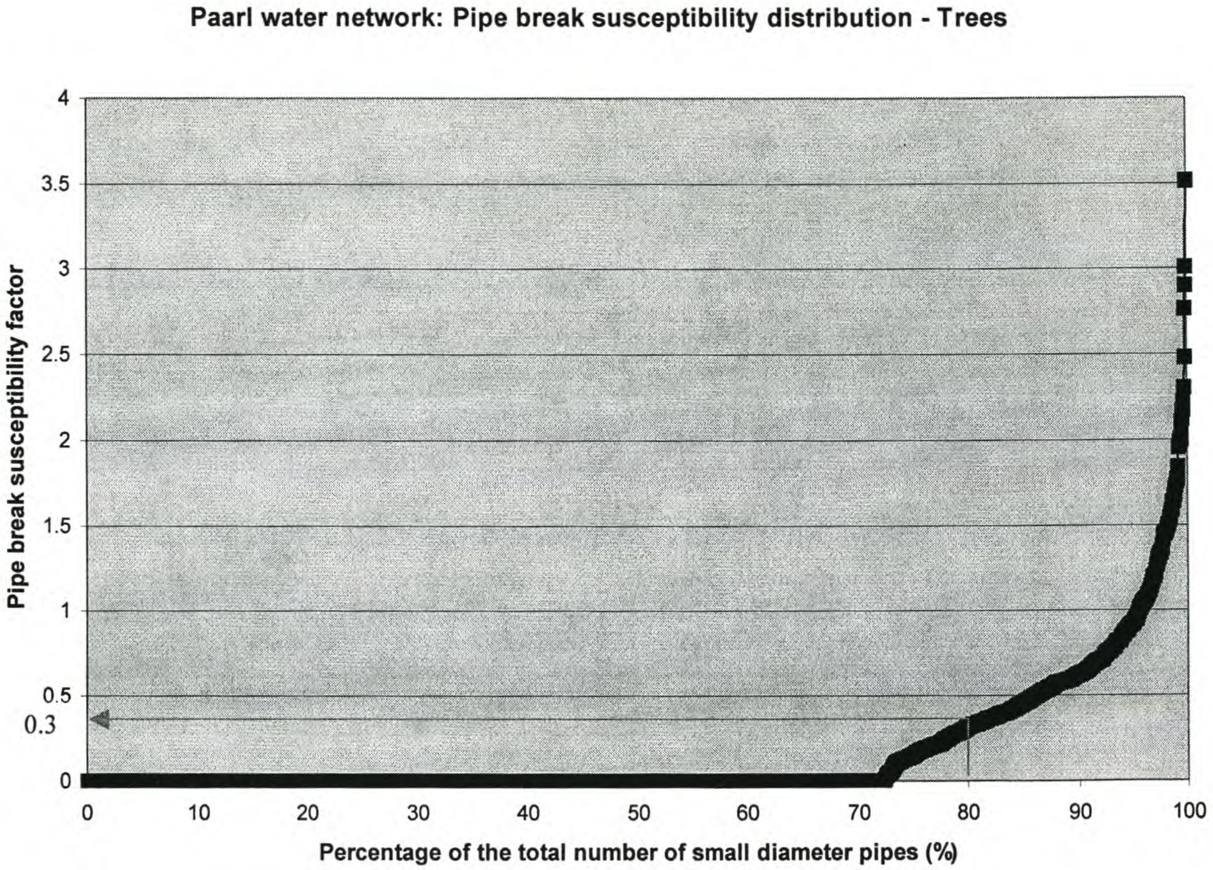


FIGURE 7.8 Pipe break susceptibility factor vs. percentage of total number of pipes with d ≤ 150 mm

7.4 PIPE BREAK IMPACT ASSESSMENT

The results of the pipe break impact assessment on the water distribution system of Paarl will be discussed in this section. The pipe break impact assessment with regard to water loss and property damage will be discussed separately under the sections that follow.

7.4.1 Water loss assessment

Since data was also available on the water distribution system layout of Wellington, and the two municipalities Paarl and Wellington fall under the same Drakenstein municipal administration (see Section 7.1.2), it was decided to conduct the water loss assessment as the result of pipe break on the joint Paarl-Wellington water distribution system.

The water loss assessment subsystem of the SDSS (see Section 6.3.1) is applied to the Paarl-Wellington water distribution system to identify pipes where *high* outflow rates can be expected during a pipe break. Figure 7.9 shows the results of an outflow analysis conducted on all 4520 pipes in the combined network. The high outflow rates in Figure 7.9 mainly occur at the large diameter pipes. Since these large diameter pipes break very seldom, a second outflow analysis (which should be more useful for preventative maintenance prioritisation) was conducted on the smaller diameter pipes only. The results of the outflow analysis conducted on the 3831 small diameter pipes (d ≤ 150 mm) in the network, are shown in Figure 7.10. An outflow rate ≥ 200 l/s can be classified as *high*, since only 20% of all small diameter pipes have outflow rates of that magnitude (see Figure 7.10). Such high outflow rates in the small diameter pipes can only occur when the break is near the connection to a large diameter water main.

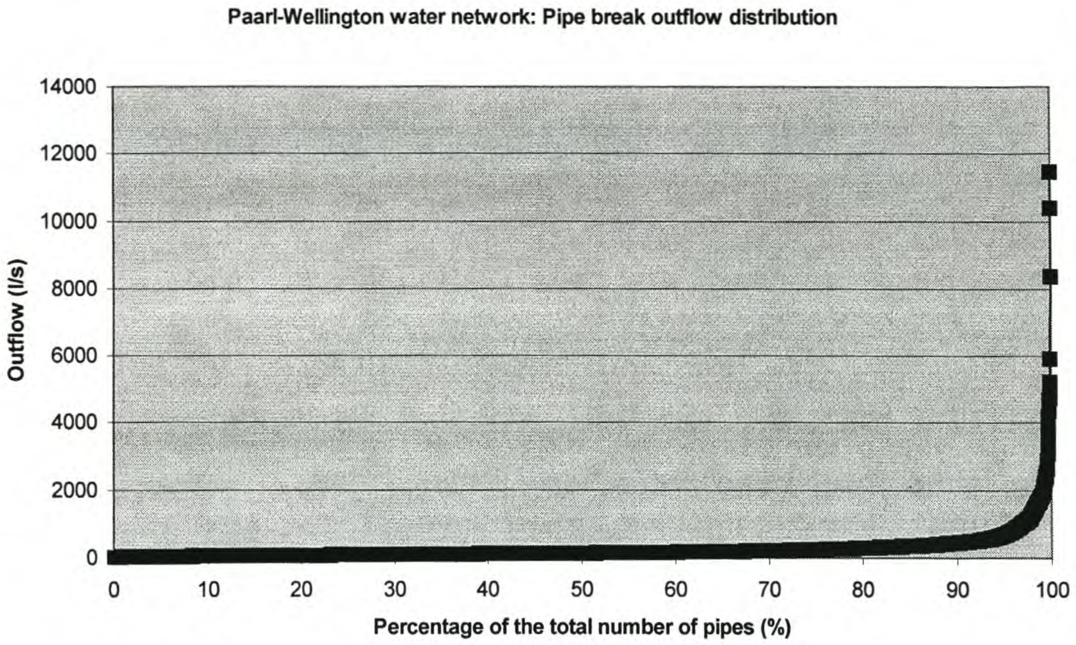


FIGURE 7.9 Pipe break outflow vs. percentage of total number of pipes

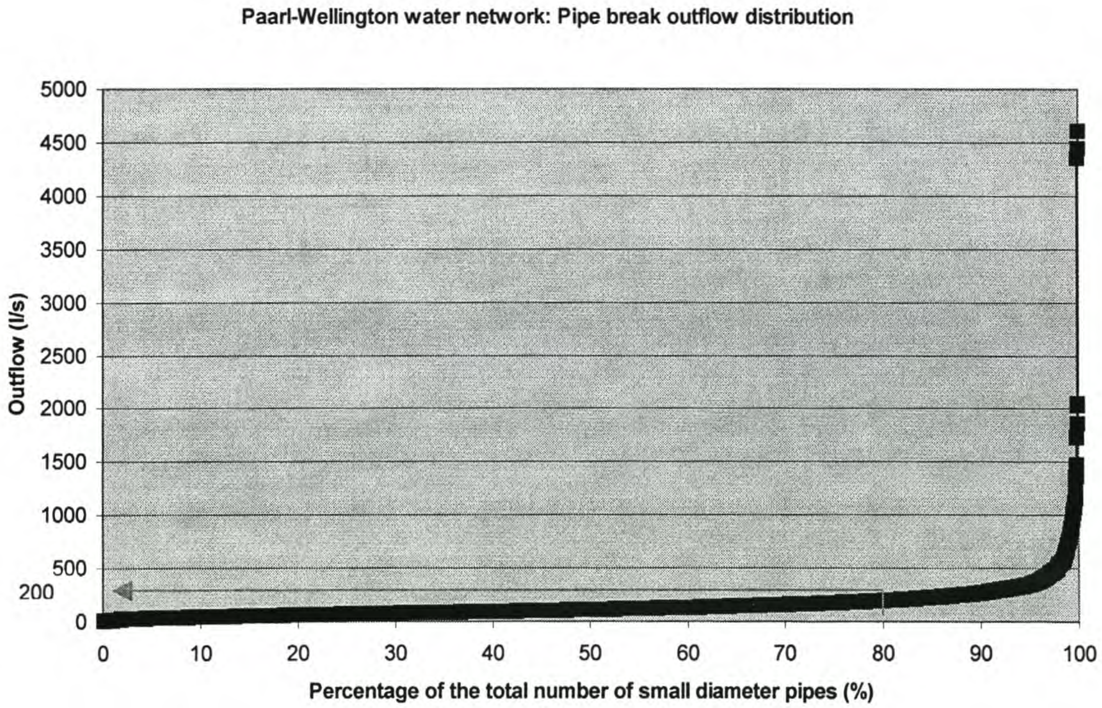


FIGURE 7.10 Pipe break outflow vs. percentage of total number of pipes with d ≤ 150 mm

7.4.2 Assessment of potential damage caused to property

The pipe break impact subsystem of the SDSS for property damage assessment (see Section 6.3.2) is applied to the Paarl water network to identify pipes that can cause water damage to nearby property in case of a pipe break.

The pipe break impact at each stand (in the residential areas), due to its location relative to the pipe, was assessed using the fuzzy controller that is linked to the SDSS (see Section 5.4.4). The stand data used in the assessment is part of the cadastral layout data (see Section 7.1.1). The fuzzy controller input variables are *Stand_slope* and *Distance* from pipe. The fuzzy controller output variable is *Stand_risk* and the following fuzzy rule base was used (see also Sections 2.2.2 and 2.2.3):

1.0 if Stand_slope is uphill then Stand_risk is very_low
1.0 if Stand _slope is moderate and Distance is very_near then Stand_risk is moderate
1.0 if Stand _slope is moderate and Distance is near then Stand_risk is low
1.0 if Stand_slope is moderate and Distance is relative_far then Stand_risk is very_low
1.0 if Stand_slope is steep and Distance is very_near then Stand_risk is high
1.0 if Stand_slope is steep and Distance is near then Stand_risk is moderate
1.0 if Stand_slope is steep and Distance is relative_far then Stand_risk is low
1.0 if Stand_slope is steep and Distance is far then Stand_risk is very_low
1.0 if Stand_slope is very_steep and Distance is very_near then Stand_risk is very_high
1.0 if Stand_slope is very_steep and Distance is near then Stand_risk is high
1.0 if Stand_slope is very_steep and Distance is relative_far then Stand_risk is moderate
1.0 if Stand_slope is very_steep and Distance is far then Stand_risk is low
1.0 if Stand_slope is very_steep and Distance is very_far then Stand_risk is very_low

The two membership functions for the input variables viz. *Stand_slope* and *Distance*, that were used in the rule base are shown below in Figures 7.11 and 7.12.

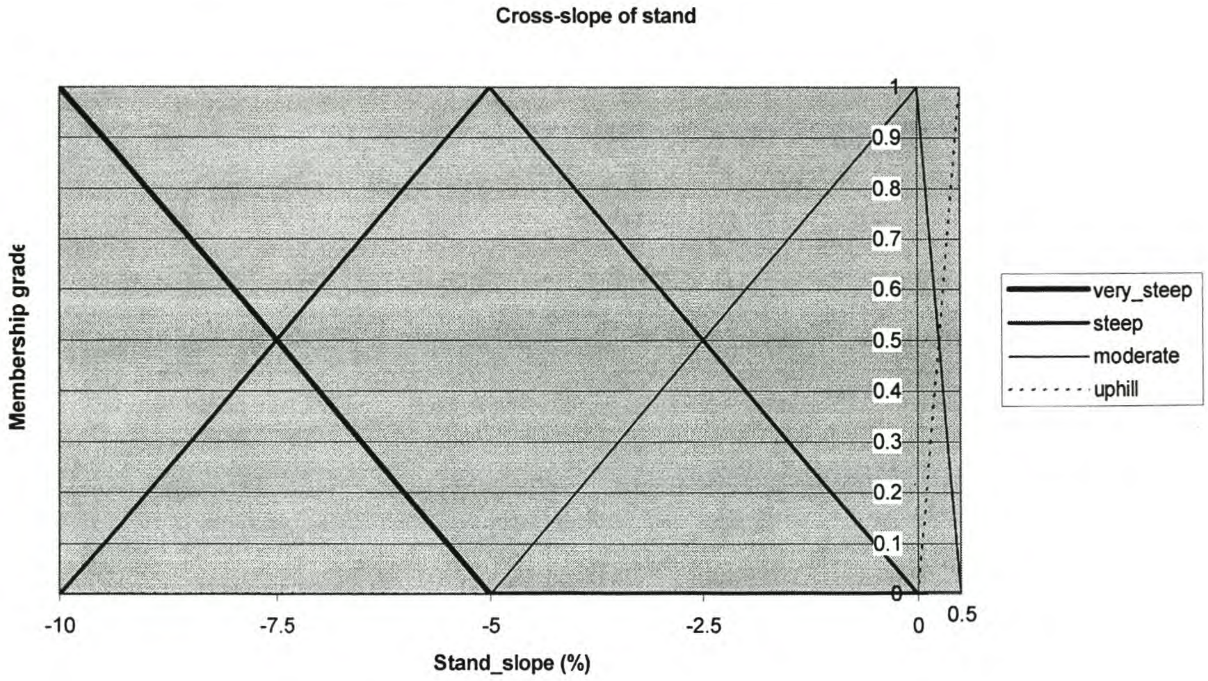


FIGURE 7.11 Membership function for input variable *Stand_slope*

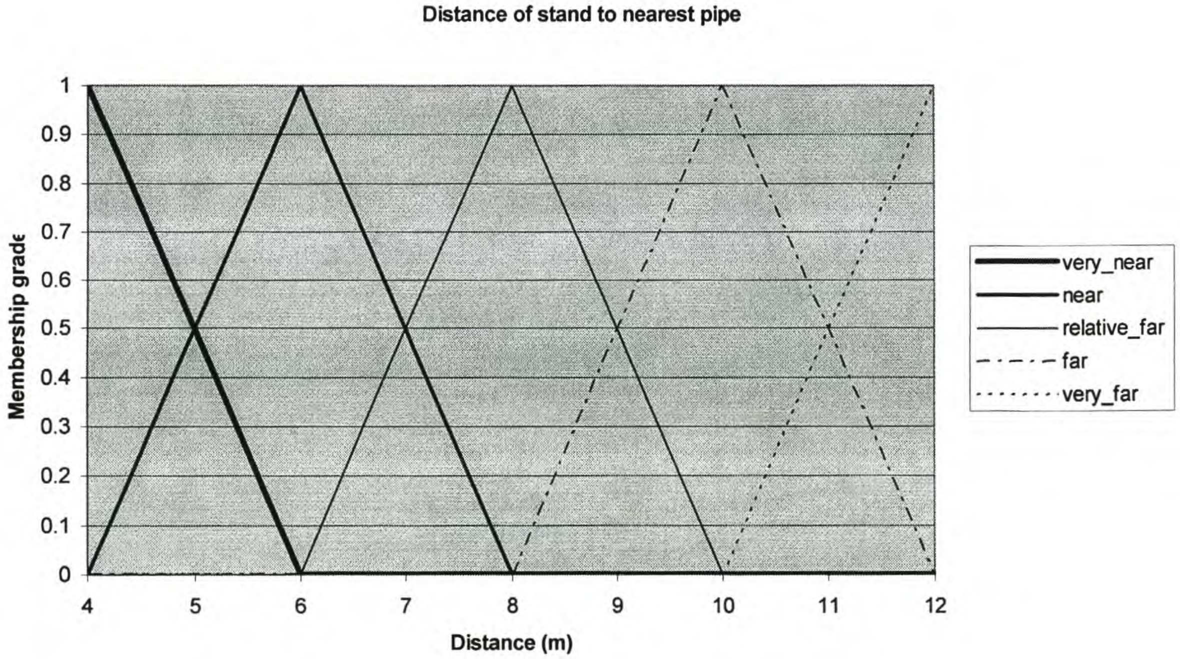


FIGURE 7.12 Membership function for input variable *Distance*

The membership function for the output variable *Stand_risk*, representing the pipe break impact at a stand is shown in Figure 7.13. The shape of the *Stand_risk* output function is not so important, since the fuzzy controller uses the simplified centroid defuzzification method (see Section 2.2.3) and only needs the centroidal x-coordinates at 0, 0.25, 0.5, 0.75 and 1. A singleton-type output member function could thus also have been specified, but the shape as shown in Figure 7.13 is more expressive.

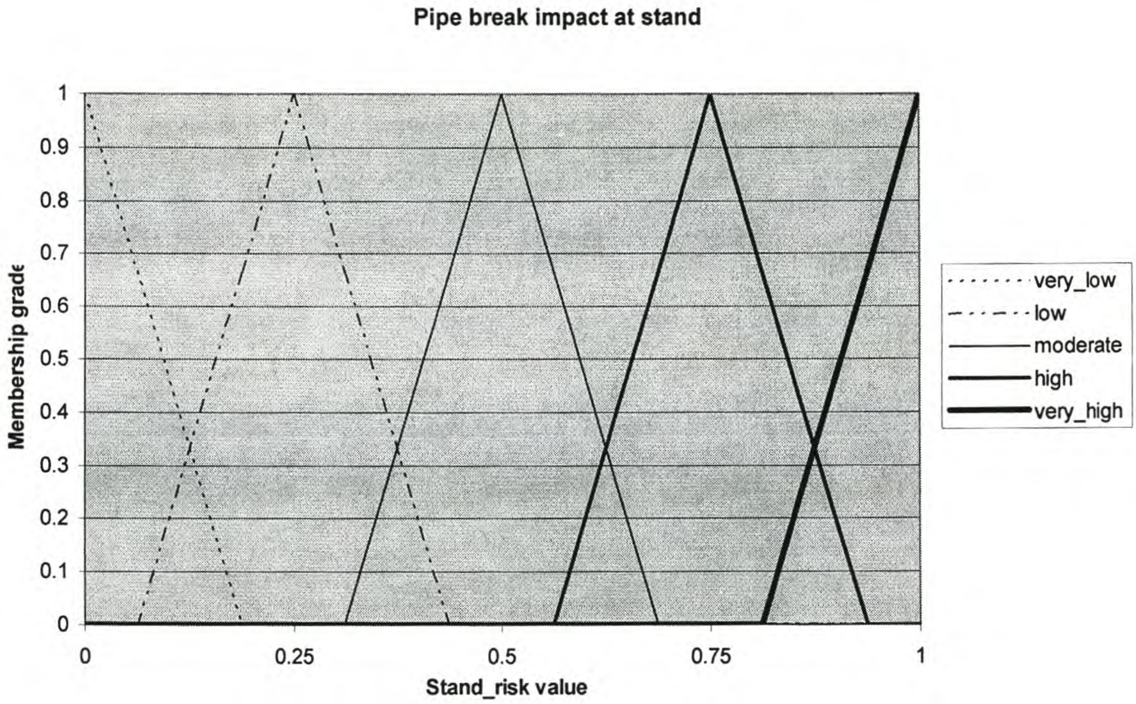
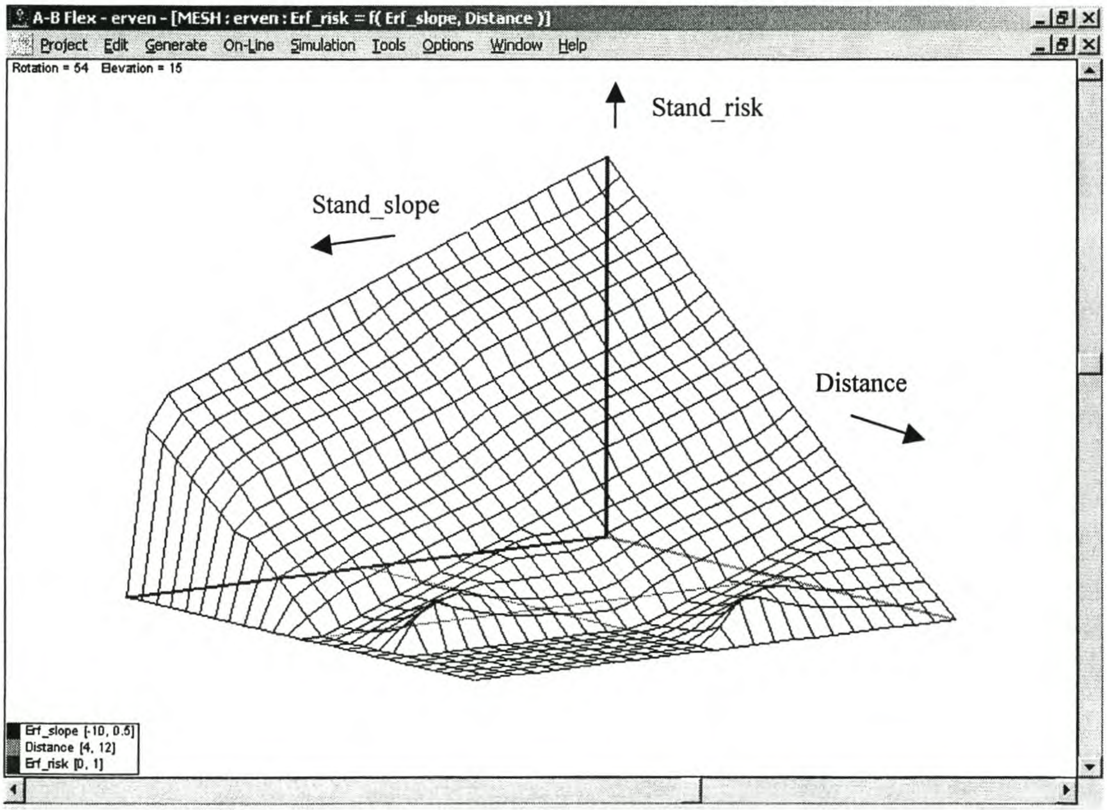


FIGURE 7.13 Membership function for output variable *Stand_risk*

The output value *Stand_risk,* which is a fuzzy function of both *Stand_slope* and *Distance,* can be displayed as a control surface as shown in Figure 7.14. As can be seen from the plot, the greater the downward cross-slope of the stand, i.e. its negative slope, the greater becomes the *Stand_risk* value. The *Stand_risk* value also increases when the distance to the nearest pipe decreases. Stands with uphill slopes (positive percentage), regardless of their distance to the nearest pipe, will all have very low *Stand_risk* assigned to them. Stands that are very far (11 m to 12 m) away from the nearest pipe, such as when the pipe is located at the sidewalk at the opposite side of a wide street, may still get a low to very low *Stand_risk* value assigned to them if the stand slopes down very steeply.


A-B Flex - erven - [MESH : erven : Erf_risk = f(Erf_slope, Distance)]
Project Edit Generate On-Line Simulation Tools Options Window Help
Stand_risk
Stand_slope
Distance

Rotation = 54 Elevation = 15

Erf_slope [-10, 0.5]
Distance [4, 12]
Erf_risk [0, 1]

FIGURE 7.14 Control surface: *Stand_risk* = f(*Stand_slope*, *Distance*)

The maximum pipe break impact of each pipe on the nearby stands was then calculated using the pipe break impact subsystem of the SDSS for property damage assessment as described in Section 6.3.2. The cumulative pipe break impact distribution of the Paarl study area is shown in Figure 7.15. The plot shows the pipe break impact factors for 2014 pipes out of the 3035 pipes in the study area. The pipes not represented in the plot are mainly those located outside the residential areas and built-up areas, where property damage cannot occur. There are, however, also some pipes excluded in the plot that are located alongside streets for which there were not enough digital data available (viz. street numbers and known elevation points for the interpolation process, see Section 6.3.2.1). A pipe break impact factor ≥ 0.58 can be classified as *high*, since only 20% of all pipes that were analysed in the study area have pipe break impact factors of that magnitude (see Figure 7.15). The *high* pipe break impact classification figure of 0.58 correlates well with the *high Stand_risk* membership function that starts at 0.56 (see Figure 7.13), which indicates that the fuzzy rule base and fuzzy membership functions suit the Paarl study area.

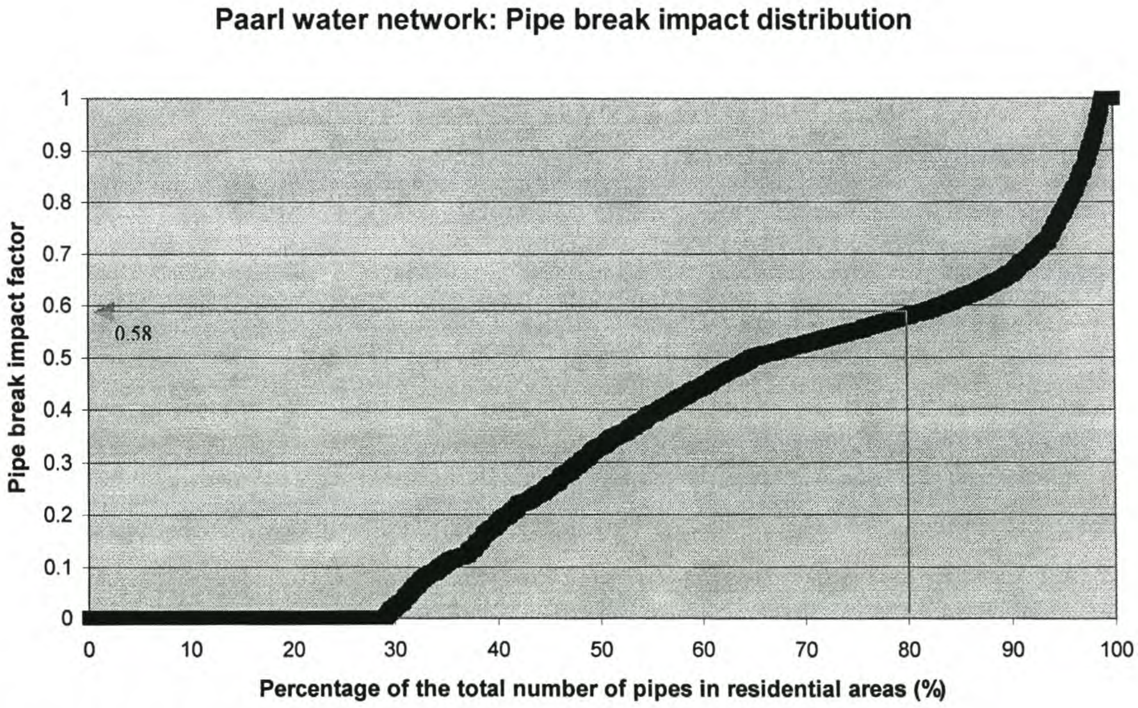


FIGURE 7.15 Pipe break impact factor – Property damage vs. percentage of total number of pipes in residential areas

7.5 COMBINED PIPE BREAK SUSCEPTIBILITY ANALYSIS AND IMPACT ASSESSMENT

The combined pipe break susceptibility analysis and impact assessment subsystem (see Section 6.4) is applied to the Paarl water network to identify pipes with high total pipe break susceptibility (as the result of age, air pockets and trees) and that have high total pipe break impact (as the result of water loss and damage caused to property).

The MFEP factor weights assigned to the various pipe break susceptibility result collections for determining the total pipe break susceptibility (see Section 6.4) are shown in Table 7.2.

TABLE 7.2 Total pipe break susceptibility categories and sub-categories

Total pipe break susceptibility categories and sub-categories	Number of pipes found	Factor weight
1. PIPES WITH HIGH BREAK SUSCEPTIBILITY - AGE		
Age ≥ 25 (years)	2127	$\frac{1}{3}$
2. PIPES WITH HIGH BREAK SUSCEPTIBILITY - AIR POCKETS		
2.1 High break susceptibility Total_airbreak ≥ 0.1	1600	$\frac{1}{3}\cdot\frac{1}{3}=\frac{1}{9}$
2.2 Very high break susceptibility Total_airbreak ≥ 0.16	790	$\frac{1}{3}\cdot\frac{1}{3}=\frac{1}{9}$
2.3 Extremely high break susceptibility Total_airbreak ≥ 0.3	91	$\frac{1}{3}\cdot\frac{1}{3}=\frac{1}{9}$
3. PIPES WITH HIGH BREAK SUSCEPTIBILITY - TREES		
3.1 High break susceptibility Tree_risk ≥ 0.5	342	$\frac{1}{3}\cdot\frac{1}{3}=\frac{1}{9}$
3.2 Very high break susceptibility Tree_risk ≥ 1.5	45	$\frac{1}{3}\cdot\frac{1}{3}=\frac{1}{9}$
3.3 Extremely high break susceptibility Tree_risk ≥ 2.0	15	$\frac{1}{3}\cdot\frac{1}{3}=\frac{1}{9}$

The cumulative distribution of the total pipe break susceptibility of the pipes in the Paarl water network is shown in Figure 7.16.

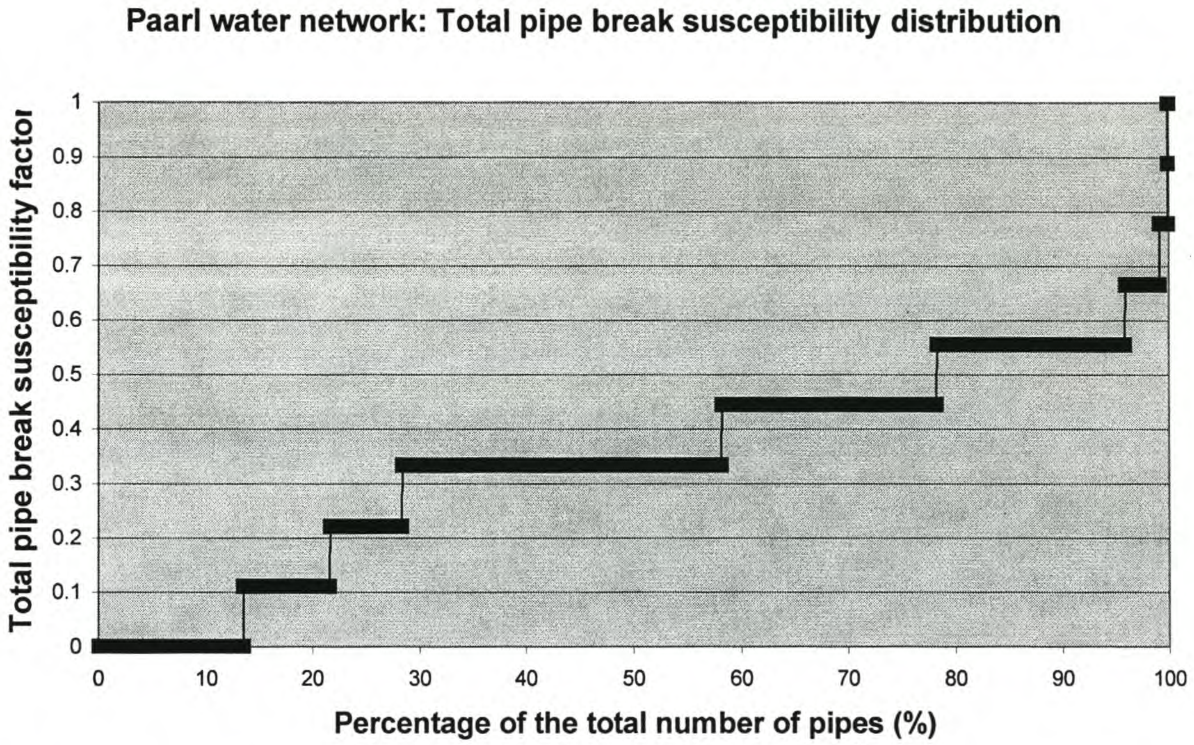


FIGURE 7.16 Total pipe break susceptibility factor vs. percentage of total number of pipes

A total pipe break susceptibility factor ≥ 0.4444 defines the *high* total pipe break susceptibility class and includes 1268 pipes out of the 3035 pipes in the study area (giving a sharpness factor of 41.8%). For a pipe to be classified as having *high* total pipe break susceptibility, it should comply with all sub-categories of one of the main pipe break cause categories (viz. age, air pockets or trees) and must at least also fall into the first sub-category of another main pipe break cause category (see Table 7.2). It was found that 103 breaks of all the 124 air pocket- and tree-related breaks in the study area were located along pipes with *high* total pipe break susceptibility (giving a success rate factor of 83.1%). The *very high* class can be defined for pipes with a total pipe break susceptibility factor ≥ 0.5555. The SDSS identified 659 pipes out of 3035 pipes that fall into this class (giving a sharpness factor of 21.7%). For a pipe to be classified as having *very high* total pipe break susceptibility, it should comply with all sub-categories of one of the main pipe break cause categories (viz. age, air pockets or trees) and must at least also fall into the first two sub-categories of another main pipe break cause category (see Table 7.2). Of all the 124 air pocket- and tree-related breaks, 74 were located along pipes with *very high* total pipe break susceptibility (giving a success rate factor of 60%). An *extremely high* total pipe break susceptibility class can be defined for pipes with a total pipe break susceptibility factor ≥ 0.6666. The SDSS identified 125 pipes out of 3035 pipes that fall into this class (giving a fine sharpness factor of 4.12%). For a pipe to be classified as having *extremely high* total pipe break susceptibility, it should comply with all sub-categories of two of the main pipe break cause categories (viz. age, air pockets or trees) (see Table 7.2). Of all the 124 air pocket- and tree-related breaks, 38 were located along pipes with *extremely high* total pipe break susceptibility (giving a success rate factor of 30.65%). The reliability factor (see Section 5.3.9) also increases as the total pipe break susceptibility classes are set higher each time, confirming that the model functions correctly.

The MFEP factor weights assigned to the pipe break impact result collections for determining the total pipe break impact (see Section 6.4) are shown in Table 7.3.

TABLE 7.3 Total pipe break impact categories and sub-categories

Total pipe break impact categories and sub-categories	Number of pipes found	Factor weight
1. PIPES WITH HIGH BREAK IMPACT - WATER LOSS		
Outflow ≥ 200 l/s; Diam ≤ 150 mm	599	$\frac{1}{2}$
2. PIPES WITH HIGH BREAK IMPACT - PROPERTY DAMAGE		
Property_damage ≥ 0.58	328	$\frac{1}{2}$

The cumulative distribution of the total pipe break impact of the pipes in the Paarl water network is shown in Figure 7.17.

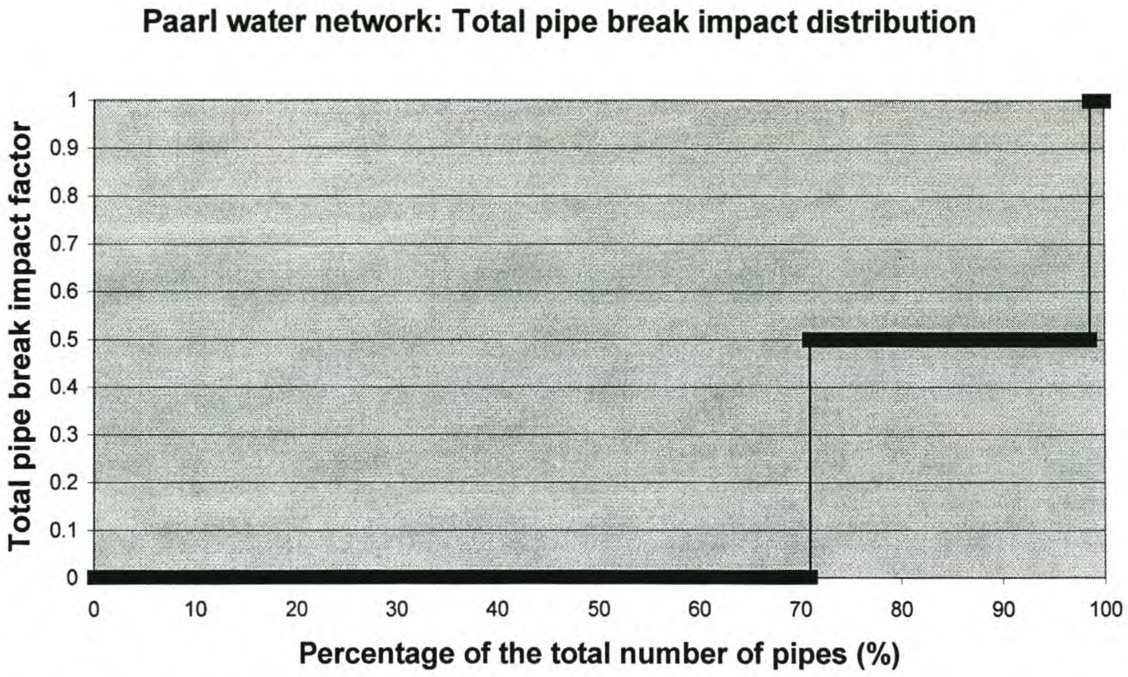


FIGURE 7.17 Total pipe break impact factor vs. percentage of total number of pipes

A total pipe break impact factor ≥ 0.5 can be specified to define the *high* total pipe break impact class (see Figure 7.17). The pipes in this class are pipes with *high* water loss potential or have *high* property damage potential during pipe break. The SDSS identified 882 pipes out of 3035 pipes that fall into this class (giving a sharpness factor of 29%). A total pipe break impact factor of 1 defines the *very high* total pipe break impact class. Pipes in this class have *high* water loss potential and *high* property damage potential during a pipe break. The SDSS identified 45 pipes out of 3035 pipes that fall into this class (giving a fine sharpness factor of 1.48%).

The final combined total pipe break susceptibility and total impact factor is then calculated for each pipe (see Section 6.4). This final results factor is made up of 50% of the total pipe break susceptibility factor and 50% of the total pipe break impact factor. The cumulative distribution of the final analysis results is shown in Figure 7.18. A final results factor (combined total pipe break susceptibility and total impact) ≥ 0.72 can be classified as *very high* (see Figure 7.18) and consists of 28 pipes out of the 3035 pipes in the network (giving a fine sharpness factor of 0.92%). The minimum requirements for a pipe to fall into this *very high* final results class can be derived from Tables 7.2 and 7.3 and would typically be the following: the pipe should have *very high* total pipe break impact (i.e. water loss and property damage) and *high* total pipe break susceptibility (as defined earlier in this section). If a pipe has an extremely high total pipe break susceptibility factor of above 0.95, then it will also be classified under the *very high* final results, if the total pipe break impact of the pipe is only 0.5 (as the result of either water loss or property damage).

Two more final results classes can be defined, viz. *high* and *extremely high*. The cut-offs for the final results factors for these two classes are ≥ 0.666 (giving a sharpness factor of 1.35%) and ≥ 0.833 (giving a very fine sharpness factor of 0.1%) respectively. The minimum requirements for a pipe to fall into the *high* final results class can be derived from Tables 7.2 and 7.3 and is slightly less restrictive than for the *very high* classification described above. For a pipe to be classified in the *extremely high* final results class, it should have *very high* total pipe break impact (i.e. water loss and property damage) and an *extremely high* total pipe break susceptibility (as defined earlier in this section).

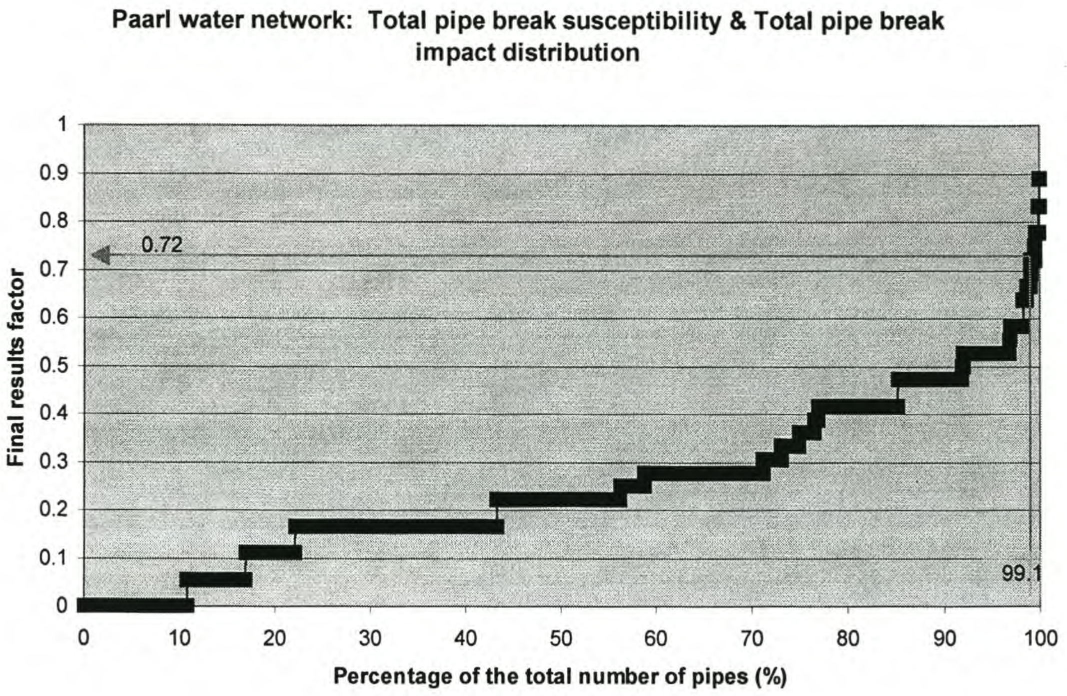


FIGURE 7.18 Final results factor vs. percentage of total number of pipes

7.6 RECOMMENDATIONS

The pipes that have *very high* final results factors ≥ 0.72 from the combined total pipe break susceptibility analysis and total impact assessment conducted in Section 7.5 can be mapped thematically with the SDSS and are shown in Figure 7.19. Very high priority should be given to the preventative maintenance of these 28 pipes, since they are susceptible to breaking and having a devastating impact as far as water loss and damage caused to property are concerned. Similar thematic maps to Figure 7.19 can be drawn for the *high* and *extremely high* final result classes. The municipal budget allocated for preventative maintenance on the water distribution system would be the deciding factor on which classification to use. There are various other combinations that can be analysed with the SDSS (see Section 6.4). Total pipe break susceptibility and potential water loss may also be an important combination to analyse and map, since water is scarce in the Paarl area - in the summer months the water supply cannot always meet the demand unless water restrictions are imposed.

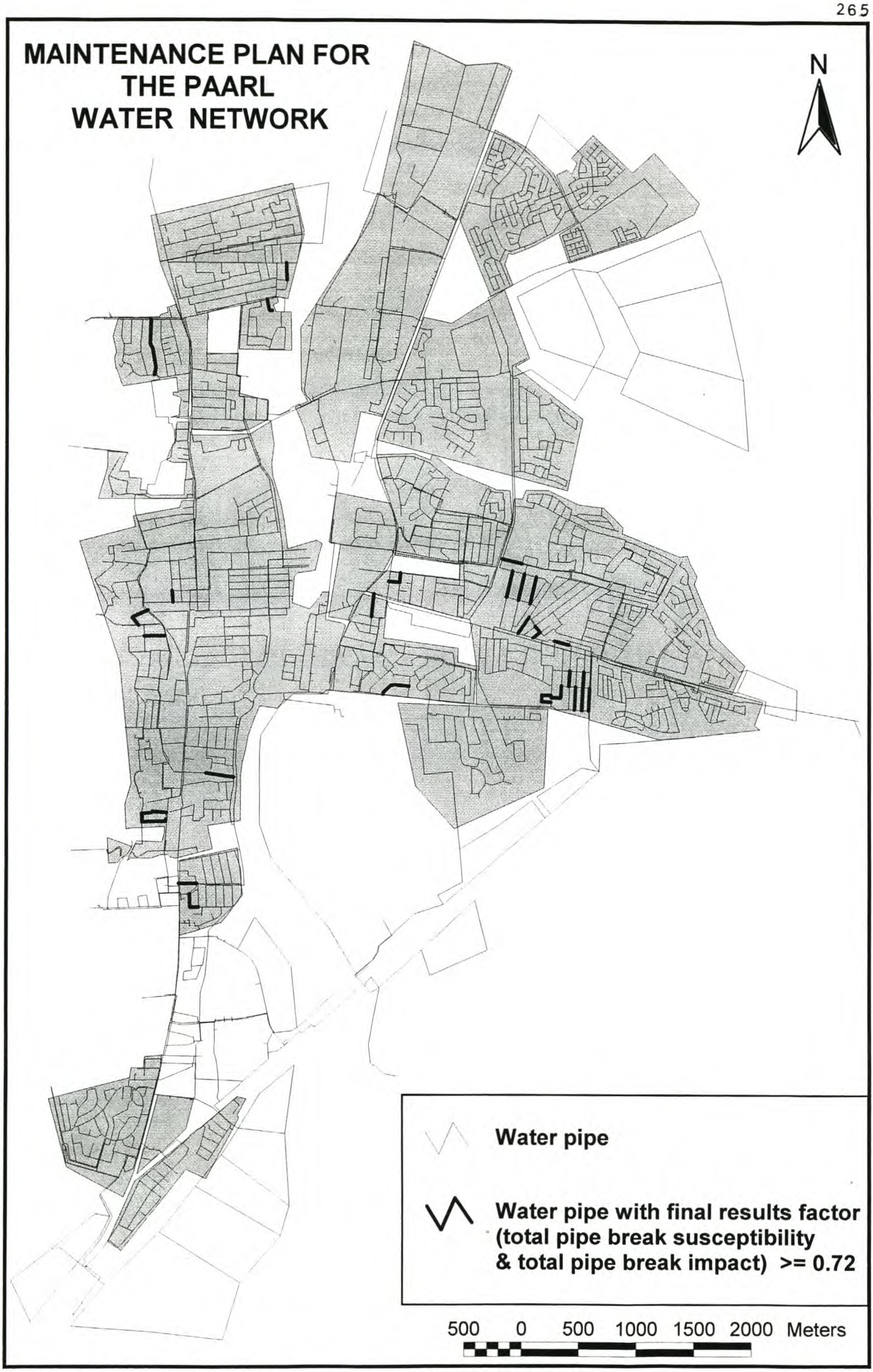

MAINTENANCE PLAN FOR
THE PAARL
WATER NETWORK
N
Water pipe
Water pipe with final results factor (total pipe break susceptibility & total pipe break impact) >= 0.72
500 0 500 1000 1500 2000 Meters

FIGURE 7.19 Pipes to receive very high preventative maintenance priority

CHAPTER 8: SUMMARY AND RECOMMENDATIONS

Municipal water distribution maintenance is very important for sustainable urban development. Water pipe breaks result not only in a disruption in service but also in significant loss of water, which otherwise could have been sold to the consumer. In countries where water is scarce, such as South Africa, water losses can be detrimental to the living standard of people. Water pipe breaks can furthermore cause extensive damage to nearby lower-lying properties.

Existing decision support systems available in the field of water distribution system maintenance are mainly focused on leak detection and pipe rehabilitation/replacement strategy. These existing systems, however, do not address the actual causes of pipe breaks and pipe break impact is also not supported.

The aim of this research was to develop a spatial decision support system (SDSS) for pipe break susceptibility analysis and impact assessment. The engineer (or public works administrator) can apply the SDSS to model the complex pipe break phenomena in the municipal water distribution system. The SDSS can identify pipes susceptible to breaking and pipes with potentially high break impact as far as water loss and damage caused to nearby property are concerned. This combined pipe break susceptibility analysis and potential impact assessment should promote more informed decision-making on preventative maintenance measures to be taken and their prioritisation.

The SDSS is based on the concepts of information systems theory and object-oriented modelling. All phases of the requirements analysis and design of the SDSS have been documented using the Unified Modelling Language (UML) diagrams. This standardised documentation approach should promote communication in subsequent system development work.

The core functionality of the SDSS is incorporated in the Object Query Browser. This Object Query Browser of the SDSS is written in Visual Basic for Applications (VBA) so that it can be fully integrated with a modern geographical information system (GIS) that has a built-in VBA Editor, such as for example ArcView 8.1. Some additional functionality is added via external programs which are linked to the SDSS:

- A fuzzy-logic-based controller (viz. A-B Flex) is linked to the system to supply the fuzzy logic analysis functionality that is needed for the modelling of complex aspects regarding pipe break susceptibility analysis and impact assessment.

- A water distribution system analysis program (viz. EPANET) is linked to the system for performing intricate network flow operations that are needed to determine the outflow rate from a pipe break.

- An interpolation program (viz. WADPOL) is linked to the system for interpolating elevation points needed in the modelling and assessment of potential damage caused to property.

- A spreadsheet (viz. MS Excel) is also linked with the SDSS to draw graphs of the analyses results.

The SDSS users can access all system functionality via the Object Query Browser. The Object Query Browser enables different user access levels and adapts the GUIs accordingly so that only the relevant user functionality is made available. The Object Query Browser furthermore provides powerful query functionality and manipulation operations. The query functions and operations can be customised at run-time in the VBA Editor of the SDSS, which provides a very flexible query environment. The query results are stored in query collections that can be queried further or can be associated with other query collections. This provides a suitable environment where the solution space can be explored. The Object Query Browser has specialised functions and operations to support the following pipe break susceptibility analyses, viz. for age, air-pocket formation and tree-root attack. The Object

Query Browser also has specialised functions and operations for the assessment of pipe break impact regarding water loss and property damage. The Object Query Browser further has functionality to combine the results from various analyses using the Multifactor Evaluation Process (MFEP) in conjunction with user-specified MFEP weighting factors. Finally, the MFEP weighting factors and also the other decision rules applied in the SDSS can further be calibrated with the Object Query Browser by evaluating the calibration statistics (i.e. factors calculated by the Object Query Browser for evaluating how well the model results correlate with actual pipe break occurrences).

The pipe break susceptibility analysis model of the SDSS is mainly based on existing pipe break theory. Pipe break theory, however, is a highly specialised field in civil engineering and literature on the topic was difficult to obtain, since only scattered pieces of information can be found in numerous articles written by municipal engineers reporting of specific pipe break causes they had encountered. In chapter 4 of this dissertation this existing, 'fragmented' information on pipe break causes has been outlined and summarised. Special pipe break susceptibility analysis subsystems have then been constructed (based on the existing information obtained on pipe break causes) to model and analyse the pipe break susceptibility due to age, air pocket formation and tree-root attack. The subsystem for analysing pipe break susceptibility due to age, estimates pipe age by the (very accessible and readily available) inauguration year attributes of the containing residential areas in which the pipes are located. The air pocket analysis subsystem analyses pipe break susceptibility as the result of seven possible types of air pocket formation that can occur in the network viz. (i) at long pipes that also have flat slopes; (ii) at pipes with blank ends; (iii) at pipes where there are changes in diameter; (iv) at nodal high points; (v) at nodal low points; (vi) at high points along the pipes; and (vii) at low points along the pipes. The subsystem for tree-root attack, applies fuzzy logic to model the effects of tree size and distance away from the pipe. The subsystem for assessing pipe break impact on nearby lower-lying properties also applies fuzzy logic in the modelling of topographical aspects such as the slope of a stand and distance away from the pipe break.

The water loss assessment subsystem calls special hydraulic functions to determine the outflow rate from a pipe. There is very little information available in the literature on this pipe break outflow modelling, especially in a network context. New pipe break outflow modelling concepts have therefore been developed in this dissertation by applying a water distribution system program in an unconventional way (the unknown variable to be determined is in this case the outflow rate and not the pressure). The modelling of pipe break impact on nearby properties as successfully implemented in this dissertation has not yet previously been attempted and requires complicate spatial analyses in conjunction with the fuzzy inferencing already mentioned. The idea followed in this dissertation to integrate a number of existing systems into a new system, is not a new idea – the final system, however, is unique and consists of an Object Query Browser which is linked via specially designed interfaces to the following existing subsystems: viz. to a GIS, a topographical interpolator, a water distribution system analysis program, a fuzzy controller and to a spreadsheet. It was found that this unique combination of subsystems offers the best platform on which to conduct pipe break susceptibility analysis and impact assessment.

The SDSS has been applied to the water distribution system of Paarl, a medium sized town in South Africa for which cadastral layout data, pipe network data and pipe break occurrence data were available. The pipe break susceptibility analysis model of the SDSS has been tested and calibrated by comparing the model results with the actual pipe break occurrence data. A combined pipe break susceptibly analysis and impact assessment has then been conducted. The total pipe break susceptibility as the result of age, air-pockets formation and trees has been determined for each pipe in the network. The total pipe break impact as far as water loss and property damage are concerned, has been assessed for each pipe. A final results factor has then been calculated for each pipe taking into account both the total pipe break susceptibility and the total pipe break impact. The SDSS has identified 28 pipes with *very high* final results factors. These pipes should receive *very high* preventative maintenance priority.

The results and new information (viz. pipe break susceptibility and impact assessment decision rules) obtained from the application and calibration of the SDSS on the Paarl water distribution system have been incorporated into the knowledge base of the SDSS to be applied later also to other water distribution systems. The pipe break susceptibility results (especially those obtained via calibration with the actual pipe break occurrences) can give a municipality valuable insights into the specific pipe break causes of their water distribution system. Preventative maintenance can then be carried out more effectively, since when the causes of pipe breaks are determined, then the correct maintenance strategy can be planned and implemented - otherwise the same mistakes will only be repeated. The pipe break rate can hereby be reduced in a network saving the municipality pipe repair costs. The results obtained from a combined pipe break susceptibility analysis and impact assessment can help to identify pipes susceptible of breaking and which will, in the case they break, also result in extensive water loss and/or property damage. Hereby water loss and also the unaccounted-for water (UFW) can be reduced in the system, which should be of utmost importance to water scarce countries such as South Africa. Pipe break damage caused to lower-lying properties (such as the ruining of swimming pools, gardens and carpets, as well as structural damage to brick walls) can also be prevented by conducting with the SDSS a combined pipe break susceptibility analysis and impact assessment. Newly planned water distribution systems can also be tested by identifying with the SDSS pipes with high break susceptibility (which will in this case mainly be the result of air pockets) and pipes with potentially high break impact regarding water loss and property damage (caused to the planned developments). In this way designs can be altered or improved before commencing with the actual construction.

The costs of implementing the system are made up of following components: data capturing costs, software purchasing costs and user training costs. The data capturing costs will mainly consist of the digital capturing of the trees and the pipe breaks. All other data such as the cadastral layout data, pipe network data, elevation points data, etc. should readily be available from the town planning and engineering divisions of a municipality where these data sets are used in town planning and in water distribution system analyses. The following software products need to be purchased: viz. the Object Query Browser, ArcView 8.1, WADPOL and MS Excel 97. ArcView 8.1 is the most expensive of these four software products. A light GIS (viewer) could be considered to be incorporated into the system in the future (instead of ArcView 8.1), provided that this light GIS has a built-in VBA editor and can also perform a spatial join operation, which is a rather complex GIS operation used in pipe break susceptibility analysis and also for pipe break impact assessment. The other software products incorporated into the SDSS, viz. EPANET 2.0 from the US Environmental Protection Agency and A-B Flex 2.50a from Allen-Bradley Co. are free of charge programs downloadable from the Internet. Finally, there are the training costs which would largely be centred around the training of the SDSS Analyst - who will have to learn almost all system functionality and who will eventually carry out the complex analyses. The other system users, viz. the Systems Administrator, SDSS Operator and Public Works Administrator would only have to learn parts of the system.

The integration of the Object Query Browser with some of the other subsystems of the SDSS could still be improved. The integration of the Object Query Browser and ArcView will automatically improve when the new version 8.1 is installed, since the Object Query Browser, which is written in VBA, can then run directly from the built-in VBA Editor of ArcView 8.1. The Object Query Browser and the water loss subsystem are already seamlessly integrated via the EPANET hydraulic DLL functions that are called from the Object Query Browser. The rather loose coupling of the Object Query Browser with the WADPOL topographical interpolation subsystem can be improved by recompiling the source code of the WADPOL interpolator to a DLL (a few adjustments to the source code will, however, have to be made). The topographical interpolator can then be called directly as a DLL function, just as seamlessly as the water loss subsystem, and data can then be transferred directly to the interpolator via the function arguments. The A-B Flex fuzzy controller source code is not readily available and the whole program will have to be rewritten to improve the integration. If such a step is to be undertaken, it is recommended that the fuzzy controller should be rewritten in VBA. This will ascertain full integration with the Object Query Browser and the source code can then also be better maintained together with the Object Query Browser source code. The link between the Object Query Browser and MS Excel should be sustained, since it will be difficult to improve on the graphing capabilities of such an established product.

The SDSS for pipe break susceptibility analysis and impact assessment, developed in this dissertation, can be extended to support more pipe break causes and impacts, including inconvenience suffered by the consumer as the result of the disruption in service. The Object Query Browser of the SDSS, with its extensions to a GIS, topographical interpolator, fuzzy controller and a spreadsheet, provides a powerful querying platform that can with slight modifications be adapted to support spatial decision-making also in other fields, such as maintenance management of sewer systems, stormwater systems, roads, telephone and electricity cable networks. These minor system modifications would mainly be to adapt the customisable built-in Object Query Browser functions and data manipulation operations to suit the specific user requirements and to offer the necessary analysis functionality for supporting spatial decision-making in the specific application field. The UML diagrams that were used in the requirements analysis and design of the SDSS, should promote communication in subsequent system development or system adaptation work.

REFERENCES

Abel, DJ, Yap, SK, Walker, G, Cameron, MA & Ackland, RG 1992. Support in spatial information systems for unstructured problem-solving. In Bresnahan, P *et al.* (eds.) *Proceedings of the 5th international symposium on spatial data handling*, Vol. 2, pp 434-443:Charleston, SC: University of South Carolina.

AC Pipes 1994a. *Hydraulic design*. (Technical Document 7A/94). Brackenfell: AC Pipes.

AC Pipes 1994b. *Product features and characteristics*. (Technical Document 3/94). Brackenfell: AC Pipes.

AC Pipes 1995. *Pressure pipe: Product information*. (Technical Document 4 9/95). Brackenfell: AC Pipes.

Adam, EE, Jr. & Ebert, RJ 1986. *Production and operations management: Concepts, models and behaviour.* 3rd ed. Englewood Cliffs, NJ: Prentice-Hall.

Allen, C 1998. Ground attack. *Civil Engineering*, *American Society of Civil Engineers* 68,1: 44-47.

Appleton, BL, Touchette, BM, French, SC & Niemiera, AX 1997. Developing a utility line arboretum. *Journal of Arboriculture* 23,6: 219-224.

Arentze, TA & Timmermans, HJP 2000. A spatial decision support system for retail plan generation and impact assessment. *Transportation Research. Part C: Emerging Technologies* 8: 361-380.

Arnold, GE 1960. Experience with main breaks in four large cities – Philadelphia. *Journal of the American Water Works Association* 52,8: 1041-1044.

Avison, DE & Fitzgerald, G 1999. Information systems development. In Currie, W (eds.) *Rethinking management information systems*, pp.245-284. Oxford: Oxford University Press.

Bazan, A 1991. Wasserverteilung. In Mutschmann, J & Stimmelmayr, F (eds.) *Taschenbuch der Wasserversorgung*. 10th eds., pp. 463-642. Stuttgart: Franckh-Kosmos.

Bennett, S, McRobb, S & Farmer, R 1999. *Object-oriented systems analysis and design using UML*. London: McGraw-Hill.

Berenji, HR 1992. Fuzzy logic controllers. In Yager, RR & Zadeh, LA (eds.) *An introduction to fuzzy logic applications in intelligent systems*, pp. 69-96. Boston: Kluwer Academic Publishers.

Böhm, A 1993. *Betrieb, Instandhaltung und Erneuerung von Wasserleitungen*. Essen:Vulkan.

Booch, G, Rumbaugh, J & Jacobson, I 1999. *The Unified Modeling Language user guide*. Menlo Park, CA: Addison Wesley Longman.

Brink, ABA 1979. *Engineering geology of southern Africa.* Vol1. Pretoria: Building Publications.

Burrough, PA & McDonnell, RA 1998. *Principles of geographical information systems.* Oxford: Oxford University Press.

Caspers, J 1994. *Object-oriented programming: Analysis, design and implementation methods*. Charleston, South Carolina: Computer Technology Research Corporation.

Chang, C 1997. *Fuzzy-logic-based programming.* Singapore: World Scientific Publishing.

Checkland, P 1981. *Systems thinking, systems practice*. Chichester: Wiley.

Checkland, P 1999. Systems thinking. In Currie, W (eds.) *Rethinking management information systems*, pp. 45-56. Oxford: Oxford University Press.

Ciottoni, AS 1983. Computerized data management in determining causes of water main breaks: The Philadelphia case study. Proceedings of the 1983 International Symposium on Urban Hydrology, Hydraulics and Sediment Control. Lexington, KY: University of Kentucky, pp. 323-329.

Clark, RM, Stafford, CL & Goodrich, JA 1982. Water distribution systems: A spatial and cost evaluation. *Journal of the Water Resources Planning and Management Division.* 108,WR3: 243-256.

Craig, RF 1987. *Soil mechanics*. 4th ed. London: Chapman & Hall.

Cross, H 1936. Analysis of flow in networks of conduits or conductors. Bulletin No. 286, Urbana Ill. University of Illinois, Engineering Experiment Station.

Davies, AG 1994. Mohale dam: Pre-engineering planning. *Magazine of the South African Institution of Civil Engineers* 2,9: 33-45.

Davis, GB 1974. *Management information systems*. New York: McGraw-Hill.

Deist, LA 1993. Fuzzy logic controller for an autonomous guided vehicle. M.Eng thesis. Stellenbosch: Department of Industrial Engineering, University of Stellenbosch.

Densham, PJ 1992. Spatial decision support systems. In Maguire, DJ, Goodchild, MF & Rhind, DW (eds.) *Geographical information systems: Principles and applications.* Vol. 1., pp. 403-412. New York: Longman Scientific and Technical.

Department of Housing 1995. *Guidelines for the provision of engineering services and amenities in residential township development*. Pretoria: CSIR, Division of Building Technology.

Department of Water Affairs 1986. *Management of the water resources of the Republic of South Africa*. Pretoria: Department of Water Affairs.

Department of Water Affairs and Forestry 1999. A major water plan for the Cape Metro: A vision for increased water availability for the Western Cape through an integrated approach. [Online]. Available: http://www.dwaf.gov.za/Projects/Capewaterplan/ [14.6.1999].

Dickerson, JA & Kosko, B 1994. Ellipsoidal learning and fuzzy throttle control for platoons of smart cars. In Yager, RR & Zadeh, LA (eds.) *Fuzzy sets, neural networks, and soft computing*, pp. 63-84. New York: Van Nostrand Reinhold.

Duffy, NM & Hough, PK 1985. *Decision support systems: A view from the top*. Braamfontein: The Centre for Business Studies, Graduate School of Business Administration (Research Paper no. 6).

Duligal, EA 1990. *Corrosion control and quality assurance*. Johannesburg: South African Institution of Civil Engineers, Division of Water Engineering, in conjunction with Fibre-Cement Association (one-day course in Pipeline Engineering).

Eaglestone, B & Ridley, M 1998. *Object databases: An introduction*. London: McGraw-Hill.

Eikestadnuus 1999. Wanneer gaan pype nie meer bars? 1 October: 6.

Eikestadnuus 2001a. Inwoners sat vir waterpype wat bars. 24 August: 5.

Eikestadnuus 2001b. Wat van ons pype in Onder-Papegaaiberg? 31 August: 6.

Emery, JC 1969. Management information systems. In Aronofsky, JS (eds.) *Progress in operations research: Relationship between operations research and the computer.* Vol. III, pp.491-524. New York: Wiley.

England, R 2000. Getting into focus with fuzzy logic. *GEOEurope*, April 2000: 20-23.

ESRI 1996. *Using ArcView.* Redlands, CA: Environmental Systems Research Institute.

ESRI 2001. *What is ArcGIS?* Redlands, CA: Environmental Systems Research Institute.

Flood, RL & Carson, ER 1988. *Dealing with complexity: An introduction to the theory and application of systems science*. New York: Plenum Press.

Frank, WC, Thill, JC & Batta, R 2000. Spatial decision support system for hazardous material truck routing. *Transportation Research. Part C: Emerging Technologies* 8: 337-359.

Goulter, IC & Kazemi, A 1988. Spatial and temporal groupings of water main pipe breakage in Winnipeg. *Canadian Journal of Civil Engineering* 15,1:91-97.

Goulter, IC & Kazemi, A 1989. Analysis of water distribution pipe failure types in Winnipeg, Canada. *Journal of Transportation Engineering* 115,2: 95-111.

Grimshaw, DJ 1994. *Bringing GIS into business*. Harlow: Longman.

Groninger, JW, Zedaker, SM & Seiler, JR 1997. Herbicides to control tree roots in sewer lines. *Journal of Arboriculture* 23,5: 169-172.

Gulick, J 1986. Solutions to sidewalk problems. *American Forests*, May 1986: 12-15.

Habibian, A 1992. Developing and utilizing databases for water main rehabilitation. *Journal American Water Works Association* 84,7: 75-79.

Hall, WA & Dracup, JA 1970. *Water resources systems engineering*. New York: McGraw-Hill.

Horstkotte, E 2001. Fuzzy logic overview. [Online]. Available: http://www.austinlinks.com/Fuzzy/overview.html [15.2.2001].

Hudak, PF, Sadler, B & Hunter, BA 1998. Analyzing underground water-pipe breaks in residual soils. *Water engineering & Management* 145,12: 15-20.

Hutchinson, S & Daniel, L 1995. *Inside ArcView*. Santa Fe, NM: OnWord Press.

Huxhold, WE & Levinsohn, AG 1995. *Managing geographic information system projects*. New York: Oxford University Press.

Jamshidi, M 1994. On software and hardware applications of fuzzy logic. In Yager, RR & Zadeh, LA (eds.) *Fuzzy sets, neural networks, and soft computing*, pp. 396-430. New York: Van Nostrand Reinhold.

Kalinske, AA & Bliss, PH 1943. Removal of air from pipe lines by flowing water. *Civil Engineering, American Society of Civil Engineers* 13,10: 480-482.

Keen, PGW & Scott-Morton, MS 1978. *Decision support systems: An organizational perspective*. Reading, MA: Addison-Wesley Publishing.

Kettler, AJ & Goulter, IC 1985. An analysis of pipe breakage in urban water distribution networks. *Canadian Journal of Civil Engineering* 12,2: 286-293.

Klir, GJ & Folger, TA 1988. *Fuzzy sets, uncertainty, and information.* Englewood Cliffs, NJ: Prentice-Hall.

Klir, GJ & Yuan, B 1995. *Fuzzy sets and fuzzy logic: Theory and applications.* Upper Saddle River, NJ: Prentice-Hall.

Kösters, G, Pagel, B & Six, H 1997. GIS-application development with GeoOOA. *International Journal of Geographical Information Science* 11,4:307-335.

Kottmann, A 1978. Ergebnisse aus Rohrschadensuntersuchungen. *Gas und Wasserfach – Wasser/Abwasser* 119: 483-490.

Kottmann, A & Schmitt, C 1980. Unerklärliche Rohrbrüche: Mögliche Zusammenhänge zwischen Luftblasen, Fließgeschwindigkeit und Rohrbrüchen in Wasserrohrleitungen . *3R International* 19: 54-61.

Kottmann, A 1984. Luftblasen als Ursache von Rohrbrüchen. *3R International* 23: 45-53.

Kottmann, A 1995. Pipe damage due to air pockets in low pressure piping. *3R International* 34: 11-16.

Krug, R 1988. *Berechnung der Fortpflanzungsgeschwindigkeit langer Luftblasen und der Schwallströmung in horizontalen und mäßig geneigten Rohrleitungen*. Munich: Institut fur Hydraulik und Gewässerkunde, Technische Universität München, Mitteilungen Heft Nr. 49.

Lescovich, JE 1972. Locating and sizing air-release valves. *Journal American Water Works Association*, July 1972: 457-461.

Londong, J & Sauer, S 2001. MSR-Konzepte für Kläranlagen: Stand der Technik. *Korrespondenz Abwasser - Wasserwirtschaft, Abwasser, Abfall* 48,6: 778-785.

Ludewig, D 1989. Druckrohrnetzberechnung. In Bollrich, G (eds.) *Technische Hydromechanik Band 2 (Spezielle Probleme)*. Berlin: VEB Verlag für Bauwesen.

Mackintosh, G 1999. SA researchers' water stabilisation breakthrough. *SA Waterbulletin* 25,2: 16-19.

Malczewski, J 1997. Spatial Decision Support Systems, NCGIA Core Curriculum in GIScience. [Online]. Available: http://www.ncgia.ucsb.edu/giscc/units/u127/u127.html [6.10.1998].

Microsoft 1997. *Programmer's Guide: Microsoft Visual Basic version 5.0*. Seattle, Washington: Microsoft Corporation.

Morell, JD 1992. Competition for space in the urban infrastructure. *Journal of the Arboriculture* 18,2: 73-75.

Morris, RE, Jr. 1967. Principal causes and remedies of water main breaks. *Journal of the American Water Works Association* 54,7: 782-798.

O'Day, DK 1982. Organizing and analyzing leak and break data for making main replacement decisions. *Journal of the American Water Works Association* 74,11: 589-594.

O'Day, DK 1983. Analyzing infrastructure conditions – A practical approach. *Civil Engineering, American Society of Civil Engineers* 53,4: 39-42.

Powell, G, Constantinides, D & Kolovopoulos, P 1996. Addressing the problems in Roodepoort's water supply system. *Urban Management* 27,11: 48-51.

Render, B & Stair, RM, Jr. 1994. *Quantitative analysis for management*. 5th ed. Englewood Cliffs, NJ: Prentice-Hall.

Roberts, CPR 1994. Large dams and water systems in South Africa. In SANCOLD (eds.) *Large dams and water systems in South Africa*, pp 3-4. Pretoria: J.P. van der Walt and Son.

Rumbaugh, J 1997. Models through the development process. *Journal of Object-Oriented Programming*, May 1997.

SAP 1997. *R/3 System: Plant maintenance: Functions in detail – PM*. Walldorf, Germany: SAP AG.

Shen, YC & Grivas, DA 1996. Decision-support system for infrastructure preservation. *Journal of Computing in Civil Engineering* 10,1: 40-49.

Skiera, B & Hennen, G 1986. Solving sidewalk problems: Part 2. *American Forests*, June 1986: 10-13.

Sodhi, J & Sodhi, P 1999. *Software reuse: Domain analysis and design processes*. New York: McGraw-Hill.

Sprague, RH, Jr. & Carlson, ED 1982. *Building effective decision support systems.* Englewood Cliffs, NJ: Prentice-Hall.

Srinivasan, A, Sundaram, D & Davis, J 2000. *Implementing decision support systems: Methods, techniques and tools*. London: McGraw-Hill.

Stolz, A 1978. Clean and potable water for Cameroon villagers. *AC Underground, Tiefbau and Canalisation*, August 1978: 15-24.

Todini, E & Pilati, S 1987. A gradient method for the analysis of pipe networks. International Conference on Computer Applications for Water Supply and Distribution. Leicester: Leicester Polytechnic.

Twort, AC, Hoather, RC & Law, FM 1974. *Water supply.* 2nd ed. London: Edward Arnold.

UNESCO 2000. Waterday 2000 - Brochure. [Online]. Available: http://www.unesco.org/science/waterday2000/Brochure.html [19.1.2000].

US Environmental Protection Agency 1999. *Programmer's Guide: EPANET version 2.0.* Cincinnati, Ohio: US Environmental Protection Agency, Water supply and Water Resources Division.

Van Robbroeck, TPC 1994. Foreword. In SANCOLD (eds.) *Large dams and water systems in South Africa*, pp 1-2. Pretoria: J.P. van der Walt and Son.

Van Vuuren, SJ 1990. *The effect of air in water pipes*. Johannesburg: South African Institution of Civil Engineers, Division of Water Engineering, in conjunction with Fibre-Cement Association (one-day course in Pipeline Engineering).

Vent-O-Mat 1995. *Points to consider when sizing & positioning air release & vacuum break valves for water pipelines*. Johannesburg: Vent-O-Mat.

Von Bertalanffy, L 1968. *General system theory: Foundations development applications*. New York: Braziller.

Water Research Commission 2000. *Report to Parliament*. Pretoria: Water Research Commission.

Whitlow, R 1990. *Basic soil mechanics*. 2nd ed. New York: Wiley.

Wilson, LC 1985. Piled foundations through alluvial gravels: Bridges on the Berg River, Paarl Valley. In Brink, ABA (eds.) *Engineering Geology of Southern Africa*. Vol. 4, pp.262-263. Pretoria: Building Publications.

Wisner, PE, Mohsen, FN & Kouwen, N 1975. Removal of air from water lines by hyrdrulic means. *Journal of the Hydraulics Division* 101,2: 243-257.

Zadeh, LA 1965. Fuzzy sets. *Information and Control* 8: 338-353.

Zadeh, LA 1992. Knowledge representation in fuzzy logic. In Yager, RR & Zadeh, LA (eds.) *An introduction to fuzzy logic applications in intelligent systems*, pp. 1-25. Boston: Kluwer Academic Publishers.

APPENDIX A: QUERY RESULTS COLLECTION FILE

```
1: "Nodes",7
2: 10,2
3: "NODEID","NPIPES","USERNODENR","NODETYPE","ELEVATION","YCOORD",
   "XCOORD","EGL","HEAD","PRESSURE"
4: 11
5: "PIPEID","USERLINKNR","LINKTYPE","USERBEGNR","USERENDNR","LENGTH",
   "DIAM","FLOW","VELOCITY","USELEV","DSELEV"
6: "Graphfinal.xls"
7: "Mapfinal.shp"
8: "Node1409",3,3205,0,119.0065,3331,3732759,164.7943,45.7873,449.027
9: "Pipe1796",3105,0,3115,3205,195,250,5.4688,.1115,119.0065,117
10:  "Pipe1851",3206,0,3205,3283,12,100,3.5666,.4544,119.0065,119.2125
11:  "Pipe1900",3293,0,3205,3181,6,100,9.0355,1.1511,118.9179,119.0065
12:"Node1439",2,3296,0,112.6,550.3003,3731437.5867,173.6,61.0,598.57
13:"Pipe1744",3004,0,2994,3296,142,100,23.53,2.9977,112.6,112.6
14:"Pipe1908",3306,0,3286,3296,80,200,23.53,.7494,112.6,112.6
```

COMMENTS:

- The line numbers in column 0 (i.e. the first column) are included here only for descriptive purposes in the comments and are not part of the file.

- The text string in row 1 and column 1 (viz. *"Nodes"*) is the name of the query results collection.

- The integer value in row 1 and column 2 (viz. *7*) specifies the position of the query results collection icon in an internal image list that contains all possible images that can be selected in the Icon Designer sub GUI.

- The integer value in row 2 and column 1 (viz. *10*) specifies the number of attributes in the main query results collection.

- The integer value in row 2 and column 2 (viz. *2*) specifies the number of objects in the main query results collection.

- The text strings in row 3 are the attribute names of the main query results collection.

- The integer value in row 4 (viz. *11*) specifies the number of associated attributes in the associated collection.

- The text strings in row 5 are the attribute names of the objects in the associated collection.

- The text string in row 6 is the name of the MS Excel spreadsheet that is associated with the query results collection.

- The text string in row 7 is the name of the ArcView thematic map theme that is associated with the query results collection.

- The data in row 8 are the object data for the first object in the query results collection. The object identifier for this first object is *"Node1409".*

- The integer value in row 8 and column 2 (viz. *3*) specifies the number of associated objects for this first query results object.

- The data in row 9, 10 and 11 represent three objects that are associated with the object *"Node1409".*

- The data in row 12 represent the second object in the query results collection and has the following object identifier: *"Node1439".*

- The data in row 13 and 14 represent two objects that are associated with the object *"Node1439".*

APPENDIX B: SOURCE CODE

Contact Stefan Sinske
tel.: (021) 886 5140
email: stefan@sinske.com

APPENDIX C: SOIL SURVEY OF THE PAARL STUDY AREA

The soil survey presented in this appendix is a summary of a combined geological and pedological soil survey conducted by the Institute for Soil, Climate and Water, Pretoria.

Paarl is situated mainly on shale and mudrocks from the Malmesbury Group with occurrence of intrusive granite from the Cape Granite Suit at Paarlberg and also at the eastern side of Paarl at the foot of the Klein Drankenstein Mountains (see Figure C.1). Alluvium derived from the Table Mountain Group is also located along the Berg River (Wilson 1985). Five different soil units can be distinguished (see Figure C.1):

i) The geology of soil unit A consists of Cape Granite with a very thin surface layer of weathered granitic soils.

ii) Soil unit B consists of highly weathered red and yellow coloured neocutanic soil. This soil contains residual kaolinitic clay derived from Cape Granite. The soil is non-expansive (low shrink-swell potential) and internal drainage can be classified as good.

iii) Soil unit C is located on the low-lying foothills. These soils may consist of hydromorphic soils of the Kroonstad and Estcourt forms. These soils have a high clay content consisting mainly of residual clay from the Malmesbury Group. These soils are moderately expansive (moderate shrink-swell potential) and the internal drainage is severely impeded. Organic material may occur in these soils, especially in hydromorphic landscape portions.

iv) Soil unit D consists of young, light textured alluvium derived from the Table Mountain Group. The soil consists of sand, gravel and a small percentage of small-size boulders. Residual clay from the Malmesbury Group is located below the alluvium, at a depth of approximately 5 metres, from where it then gradually changes into soft rock shale (Wilson 1985). Expansion cannot occur in the sandy, gravelly topsoil (low shrink-swell potential) and the drainage is also good.

v) Soil unit E consists mainly of Malmesbury clay. On top of the Malmesbury clay can in some places alluvial material be found, which dates from an older period when the Berg River flowed through this region. The Malmesbury clay is moderately expansive (moderate shrink-swell potential) and internal drainage can be impeded. The intermitted alluvium sections, however, are non-expansive (low shrink-swell potential) and well-drained. This binary (duplex) nature of the soil may cause differential settlements in the topsoil at the transitional sections where the soil changes from clay into the alluvium.

The five soil units described above, and depicted in Figure C.1 have fuzzy boundaries with an estimated overlap of 200 m (extending in both directions of a boundary line). The fuzzy boundaries are inevitable, since crisp boundaries do not occur in nature. Detailed soil investigations (scale 1 : 2500 or larger) can, however, reduce the overlap to obtain sharper fuzzy boundaries.

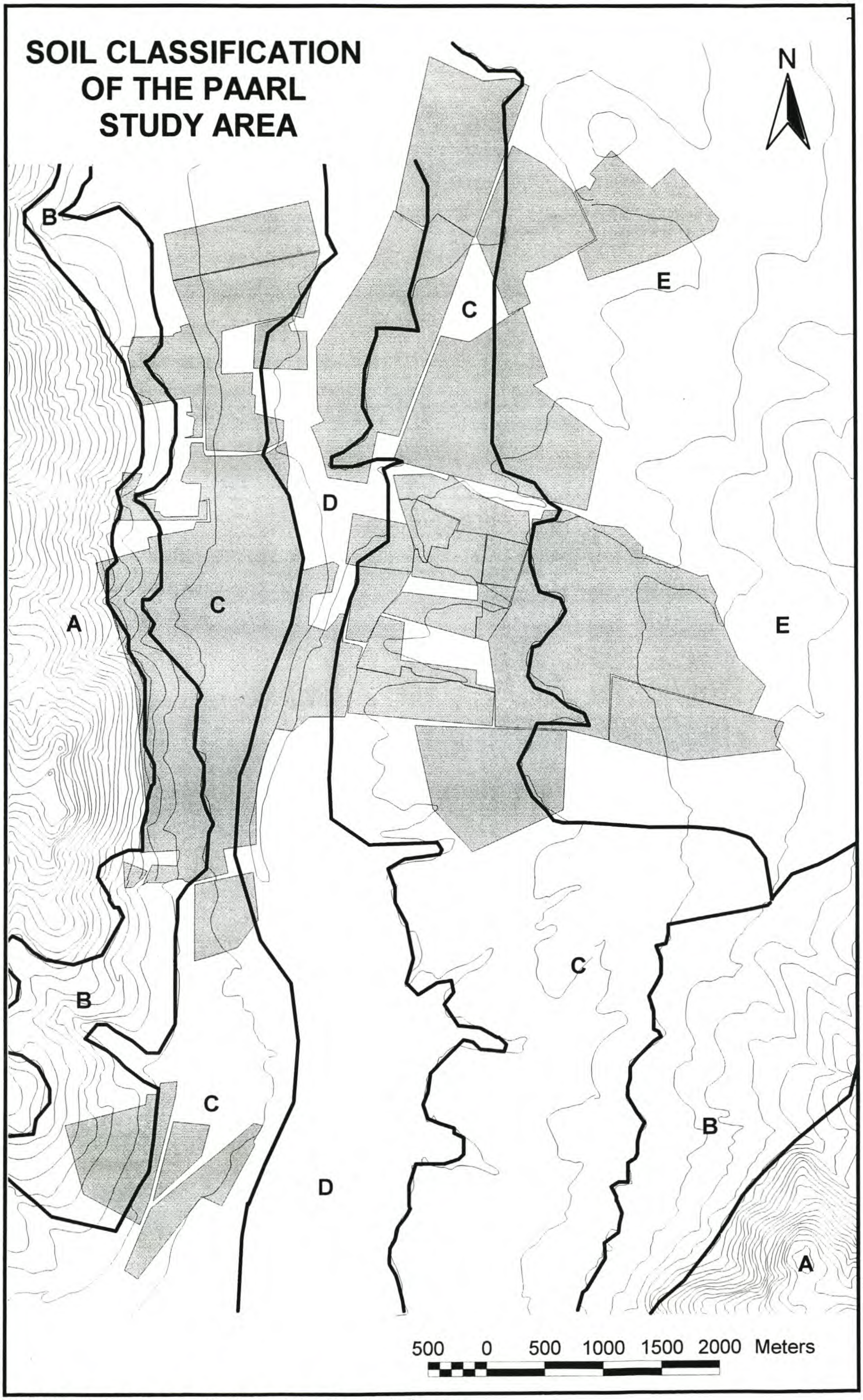


FIGURE C.1 Soil classification of the Paarl study area